Forth in Computer Systems Engineering Education

C.A.Maynard B.Sc.(Manc), M.Sc.(Kent), SMIREE, MIEEE

Head, Department of Computer Engineering,
Curtin University of Technology, Bentley, W.A. 6102
and
Wave-onic Associates, PO Box 607, Cannington, W.A.6107

Synopsis:
This paper discusses the areas of application and the reasons for the choice of the Forth language in the training of Computer Systems Engineering students at the Curtin University of Technology.

Introduction:

During the last four years there has been a significant growth in the use of the Forth language in the training of students within the School of Electrical and Computer Engineering at Curtin University of Technology and in particular within the Computer Systems Engineering degree. The variety of applications, and the reasons for the introduction of Forth are presented to show the potential value of this language within engineering education.

The Original Assignment:

The first introduction of Forth at Curtin was as a student assignment. A major assignment was initiated in 1984 to investigate the language, its uses and to suggest applications appropriate to computer systems engineering courses. The language implementation available at that time was for the Z-80 in a CP/M style development environment(CDOS).
The results of this investigation showed that the most likely applications for Forth were:

1) in electronic/computer systems projects in the hardware area and in relation to industrial control.
2) in the teaching of assembly language programming.
3) in the teaching of certain operating system concepts.

It was noted that many of the students became quite interested in Forth during the assignment and had taken the investigation significantly beyond the level necessary for the unit. Discussions showed that the students had really enjoyed using the language particularly when compared to their reaction to some other software requirements.

Implementations of Forth:

Four distinct commercial implementations of Forth have been used in Computer Systems Engineering course at Curtin and reflect the division between IBM/PC users (and clones) and Macintosh users with an approximate 50-50 split between staff users and also in the student laboratory facilities.

1) LMI PC/Forth (and UR/FORTH) for the IBM/PC etc.
2) MacForth for the Macintosh
3) New Micros MaxForth for 68HC11 single chip microprocessor
4) Mach II for the Macintosh.

Projects:

The use of Forth in undergraduate student projects, which last through the final eighteen months of the engineering course, has grown with staff familiarity. The first three implementations mentioned have been and are being used for project developments with varying degrees of success.

Forth is now also being used at the postgraduate study and research levels.

LMI Forth for the IBM/PC.

Forth for the IBM/PC and clones has been used for several projects.

One of the current projects is the development of a prefix (or standard) notation assembler to improve the application of UR/FORTH for student introduction to the assembly language for the 8086. Whilst the Forth "purist" may not support this development it provides significant advantages in teaching. This project complements the facilities in Mach II which are discussed later in this paper under the teaching of assembly language.

The latest at the postgraduate level is the development of improved industrial control software for heavy industrial application.

MacForth for the Macintosh.

This product appeared very useful when first obtained, in particular the interactive tutorial introduction to Forth was enjoyed by the students. The only successful project which has been undertaken using this version of Forth has been the implementation of a control programme for a small demonstration robot system which could be provided with command strings via an RS232C serial data link. The creation of independent commands was implemented with the Macintosh dialog interface and "radio buttons" together with editable text insertion. This project progressed to the level where a command implemented movement on three axes of the robot for each command string constructed. Most problems where experienced in the user interface aspects of the project and in the reliability of hte demonstration robot arm.

Forth on the Single Chip 68HC11 Microprocessor.

A number of projects at Curtin have been involved with the development of ROVs (Remote Operated Vehicles) for underwater application. The newer ROVs are much smaller than their predecessors to cut down power requirements and to be more manouverable. The first AUSSIE submersible produced used an STD bus Z-80 based system both at the surface and below for communication and control purposes. To help reduce the volume of electronics the use of single chip microprocessor controllers had been advocated but not implemented until the software development facilities could be made available. Late last year the suggestion was made that a single chip microprocessor with a built in Forth could implement the required functions with minimal development time and facilities. The resultant use of the MaxForth implementation on the 68HC11 is proceeding very satisfactorily and being configured to implement the communications and control functions both at the surface and within the ROV itself.

Teaching Assembly Language:

For a number of years the Z-80 assembly language was used extensively within the school but the advent of IBM/PC facilities and the use of Motorola 68000 processors in VME bus systems for robotics, and real time system applications has seen a significant reduction in the use of the Z-80.

The need was to introduce assembly language as efficiently as possible. Once a student has become familiar with the concepts then conversion to the specifics of an alternative processor seems to be less of a problem. Forth was considered as

an interactive learning environment in which access to assembler and debugging was extremely easy, however, for some time this was rejected simply because of the traditional Forth Reverse Polish Notation approach to assembler. The idea of introducing students to the RPN syntax and then to say that all normal assemblers behave differently was just not acceptable.
This situation has changed with the availability of Mach II on the Macintosh. This product has now been used in three separate units at the undergraduate and graduate diploma level to teach assembly language programming for the 68000 processor. The results have been very encouraging. In effect the students are learning two languages at once but this does not appear to cause any problems. The debugging and trace facilities allow the students to quickly relate Forth to assembler and to understand the use of macros definitions to generate inline code. Possibly of equal value, and not part of the Forth-83 standard, are the concepts of local variables and named parameters. The Mach II implementation provides the students with very obvious and understandable examples of stack frames. Consequently it has been found that the students can easily relate to the procedure linking requirements of other high level languages such as Pascal, Modula-2 and C with which they are familiar. I believe the development of a "conventional" assembler for Forth implementations on the IBM/PC could open up wider educational markets for these products. This has resulted in the current undergraduate project to implement this feature.

Multitasking Concepts:

Teaching multitasking and real time systems for engineers has generally been an onerous task. The school for some years used a real time executive for the Z-80 from Kadak in Canada which was used both for teaching and project application. Other facilities used include Versados and OS-9 for the 68000. A simpler learning environment was desireable when dealing with the operating system principles and then allowing specific implementations to be used in real time applications. The concepts of task switching, task communication etc needed to be presented in a direct learning environment. With some misgivings because of its round robin non pre-emptive task switching configuration Mach II was used as the multitasking learning tool and with emphasis on its limitations an important part of the study process. The students using Mach II for this process had already become familiar with it through the assembler studies.
A combination of lecture and assignment were used to get students to face up to

the study of a multitasking environment. The results were quite encouraging. Assignments received have included .

1) a simulation for a beer bottling plant in which semaphore systems controlled the conveyor etc in response to availability of bottles, beer and the status of the "beer store".

2) a replacement for the standard scheduler in Mach II with a priority based system together with an effective demonstration of its functionality.

New Developments

The school has just purchased a UMI robot from the UK which is programmed in a version of Forth from an IBM/PC. The "RTX" teach package for the robot is an application layer built on top of an 83-standard Forth system. The movements of the robot axes may be directly controlled by single keystrokes on the keyboard using Function keys and the numeric keypad area.

Forth words have been defined to set and read the control parameters of the robot together with words which control a learning mode give considerable flexibility to system configuration.

This is adding considerably to the appreciation of Forth as a practical language for use in the industrial control field.

Conclusion:

The application of Forth to the training of engineering students (specifically Computer System Engineering students) has grown quite dramatically during the last four years simply because it suits the design principles and systems requirements within the course. It is anticipated that Forth will continue to prove to be a significant factor in the training programme because of its easy learning characteristics and its simple access to the hardware through assembly language.

The availability of small industrial robots configured and controlled by Forth based commands will further enhance its application to a broader spectrum of engineering students.

TEACHING FORTH

Feisal Ramadan
North Sydney College of TAFE
School of Electrical Engineering
GORE-HILL,N.S.W. 2065
AUSTRALIA

1 BACKGROUND

1.1 COLLEGE

Software Engineering Forth is the name of a post certificate course offered at North Sydney College of TAFE (NSCT) School of Electrical Engineering.

The school is a government tertiary education institution with branches all over the state . Its students are mostly high school leavers. They graduate after 2 1/2 years of full time study and are awarded a certificate . Part timers take 4 years or longer . Some of the graduates are accepted and proceed to institutes of higher education or universities .

At the moment undergraduates in the digital strand are offered among other subjects M/C language programing for 2 CPU's : the 6502 for beginners in stage three followed by an advanced 6502 and/or PDP-11 for the final stage 4. In the advanced 6502 course students are encouraged to build their own hardware and write interrupt driven software to control I/O peripheral chips. The software is burned in Eprom. Until this year BASIC was the only available high level language.

Participants to the FORTH course come from the electronic industry and have as a minimum prerequisite a certificate in Electrical and Electronic engineering or equivalent qualification. Some are Institute graduates. Participants' ages vary from 22 to 55 , they come from the private sector ,government and government authorities e.g. WATERBOARD, CSIRO ,ABC ,DEFENCE , SCC...etc. Most are experienced in testing and building electronic equipment and are hardware oriented but keyboard experience among them varies . Most of them have it from programing in BASIC many from programming the 6502 or the Z80 and others the PDP-11.

The Forth course is free but the cost of books , software and hardware is born by the participant. Attendance is 3 hours per week for 17 weeks. Of the eight groups who have attended the course so far 3 groups were released by their respective employers to attend day time sessions , the remaining 5 groups attended evening sessions starting at 6.30 PM.

Motivation varied from the need to assess and catch up with new software technology to the premise of efficient hardware control.

Thus the participants within and between groups are a mixture of widely divergent interests and experiences.

1.2 FORTH

There is an ongoing discussion among staff at NSCT as to the purpose of teaching software subjects in the School of Electrical Engineering. Though staff agree that it is a mixture of data acquisition , processing and display , modeling, problem solving and the control of hardware they disagree on the amount of emphasis to be placed on each component but a trend has emerged emphasizing in our School the control of hardware.

In early 1985 after experimenting with the PDP-11 fig FORTH model by J.JAMES on a PDP-34/A and LSI-23 the idea germinated of using forth in digital and software courses but the available hardware at that time was not enough to carry out the anticipated task. The PDP-34/A was accessible only on campus and on a restricted time basis . Only 4 LSI-23s with disk drives were available. Moreover many staff members shared the view that Forth is an unknown quantity and not in the main stream of things.

The breakthrough occurred in mid 1985 when 3 events occurred simultaneously and reinforced each other. NSCT acquired 15 Apricot computers running under MS-dos . The 6502 Rockwell Forth chip with a support kit was commercially available and Sydney FIG provided the public domain version of Laxen and Perry (L&P) F83. With scarce funds available for the purchase of software the L&P version of Forth was ideal in many respects. NSCT did not have to worry about distribution problems and copyright infringements , an identical version of L&P is available for Z80 computers owned by many of our students , the source code of the language is available and fully documented , technically it is robust and above all blessed with a metacompiler.

NSCT distributes the software at cost price and in the spirit of FIG.

In February 1986 NSCT enrolled the first Forth group . In July 1986 12 staff members attended an 8 week 3 hr/week Forth course . In February 1988 this year the 8th. Forth group was enrolled.

Though students experimented with RSC-Forth on the Rockwell chip and Bryte Forth on the Intel 8051 controller there was never a formal practical lab work. Students bought hardware kits late in the course and experimented with them according to their own field of interest.

Earlier this year NSCT bought Inner access Forth for the Zilog Super8 chip. It is now NSCT's adopted hardware chip. It runs at 20 MHZ and the software is identical to L&P . Its price is well within the means of students. With the aid of the metacompiler application are burned in eprom. NSCT has acquired 2 Maestro boards with the NOVIX chip to demonstrate and experiment with the latest in Forth technology.. Twice hand robots borrowed from the School of Mechanical Engineering school were programmed via serial ports.

In the last 2 years we tried three textbooks: Mastering Forth by Anderson and Tracy , the second edition of Starting Forth by Leo Brodie and Forth: A Text and reference by Kelly and Spies. All excellent textbooks for beginners though the latter goes into more depth. Students are advised to read chapter one of Brodie's Thinking Forth.

2 APPROACH TO TEACHING

2.1 A SESSION

The Forth lesson itself is an integrated blackboard/keyboard session. Each student has his own Apricot computer. The maximum number of students is 15 per class. As Forth theory is developed the student is immediately requested to test his understanding with a specific example on the keyboard. Whenever possible examples are chosen to have a visual and audible impact. Theoretical explanation never exceeds 15 minutes . Thus a student shuttles between his desk and keyboard around 10 times in a single lesson at the beginning of the course but spent more time at the keyboard as the course progresses.

In every session a set of notes is given to the student to complement the textbook material or to emphasis some point. At the beginning of the course the student purchase at cost price the L&P Forth version modified with a screen editor. The cost of the floppy is $5 . A similar version is available for those who have IBM's or compatibles.

2.2 ASSESSMENT AND AWARD.

Assessment is based on an end of course test weighing 20% of the course marks. 80% is devoted for yearwork assignments and lab tasks. Students must attend at least 80% of the 17 weeks. Successful students are issued with a letter confirming attendance and success i.e. a statement of attainment. Units are not awarded.

3 FORTH SYLLABUS

3.1 FEEL FOR LANGUAGE AND MOTIVATION.

In this first week we stress the fact that Forth is like a spoken or written language. Spaces or gaps must exist between words. The language is a dictionary of words grouped into different vocabularies. The dictionary size is increased by inventing or creating new words.

Why forth is called a threaded language . The 4 threads of L&P and the hashing of it.
The control of VDU and cursor. AT, DARK , BEEP , HELLO .
Inventing new words using old ones. The renaming and FORGETting of definitions .
Strings. Positioning of strings .
The data and return stack.
Dot and EMIT .
Delays using MS .
Changing Radix number system.
Typical examples:

```
0.   WORDS, VOCS.
1.   HELLO CR HELLO BEEP
2.   : MY-NAME-IS  ." John Simpson    " CR  CR BEEP ;
3.   DARK 30 20 AT MY-NAME-IS 200 MS BEEP  5 TIMES (LOOPS)
4.   : MON-NOM      MY-NAME-IS     ;   (French)
5.   FORGET MY-NAME-IS
6.   12 HEX  CR . 12 DECIMAL  CR .
```

3.2 VARIABLES AND CONSTANTS

CONSTANTS.
Choosing names for constants.
VARIABLES. A lot of time is spent manipulating variables.
It is a fundamental concept to Forth.
Variable is a 2 byte storage area.
It is an address of an array.
It is a system variable.
Fetch and store data.
Fetch and store a byte.
System Variables. BASE of the Forth System .
PRINTING ON.
WARNING OFF.

3.3 LOOPS AND TABLES.

DO ... LOOPS.
Loops have been tackled in the first week using TIMES.
The topic is most interesting when the loop index I is
combined with: . , U. , .R ,U.R ,D. , UD. ,ASCII and EMIT.
Print of decimal/hex , decimal/binary and ASCII tables.
Nested loops and LEAVING loops .
Introduction to the screen editor.
Zilog Super8 hardware kit.
Organization of hardware project.

3.4 EDITOR AND FILES.

Screen editor.
Editor commands. Help menu. Shadow screens.
Forth files.
Creating and Opening files.
Copy and Convey of screens.
Loading screens. Emptying dictionary.
Line skip and screen skip.

3.5 DECISIONS.

Forth flags. CPU flags.
KEY and KEY? .
IF ... THEN and IF .... ELSE ... THEN .
Nesting of control constructs. The use of KEY and KEY?
within control constructs.
Exit from a colon definition.
Strings: ." , " , .(
Positional case statement. (best case statement)

3.6 MOVING DATA.

This is one of the most "visual" sessions , most effective
and most interesting.
Arrays are created , blanked , erased and filled then
typed.
Data is moved between arrays.
Data is fetched and stored from within arrays.
Data from 1024 size arrays is updated and flushed to disk.
Blocks from COM and HEX files are fetched from disk into
RAM, manipulated, then stored back into RAM.
Memory is DUMPed.

3.7 DEBUGGING.

SEE. VIEW. DEBUG a Word.
VOCABULARY. ONLY ALSO SEAL DEFINITIONS.
Threads again.
Header of a Forth word. Fields in a header.
HIDE and REVEAL.
RECURSE. RECURSIVE.
n! . Flip-Flop examples.

3.8 PORTS: MUSIC AND ROBOTS.

Port v/s memory.
Fetching and storing data from and to ports.
Programming a 4 channel sound chip (TI) via a serial port.
Truncation , masking and double numbers.
Programing a dumb robot via serial port (when available).
NOVIX demo on Maestro Board. Speed NOVIX v/s IBM .

3.9 EXPECT AND QUIT.

EXPECT, SPAN and relatives.
Outer interpreter. QUIT.
Command line interpreter. >IN and DONE?.
Keyboard and disk interpretation.
MANY , TIMES as examples.
Vectoring the input word source.
EVAL. (as in Micromotion)

3.10 COMPILATION.

Comma , C-comma and Literal
Compilation of different constructs.
Error checking in compilation. Stack depth change.
CREATE.....DOES>

3.11 M/C LANGUAGE 1

8086 architecture.
Instructions.
M/C v/s Forth flags.
Simple M/C language routines. CODE. NEXT. END-CODE
MS-DOS interrupt calls.

3.12 M/C LANGUAGE 2

Labels. Subroutines.
Branching and jumping.
Structured conditionals.
Mixing high level low level modules .The Laxen Trick.
Inner interpreter.
Exit via R> DROP.

3.13 HARDWARE 1.

Zilog Super8 chip architecture.
Instruction set.
Zilog interrupts.
Hardware project.

3.14 HARDWARE 2.

Novix chip Architecture.
Instruction set.
Hardware project.

3.15 HARDWARE 3.

Novix optimiser.
Hardware project.

3.16 MULTITASKING

Round robin multi tasking loop. Pause.
User variable and system variables area.
Task creation . Task wake and sleep. Background task.
Single and Multi.
A multi-task example is demonstrated.
Hardware project.

3.17 METACOMPILATION

The meaning of metacompilation.
The mechanics of metacompilation.
Metacompiling a headless application.
Finalizing hardware project.
NOVIX metacompilation.

4 GENERAL COMMENTS

Though the source code of L&P is available and fully documented in the shadow screens no beginner can start using it efficiently without external help in the form of extra notes or a demonstration . This is absolutely no reflection on the L&P version far from it. It is just that the documentation in L&P is not a tutorial on the Forth Language.

The massive source code initially bewilders the beginner and opens up a pandora's box of questions which can sidetrack the instructor. But ultimately it is for the serious student a gold mine of information on Forth programming techniques , topics , style and tricks. Some words which initially confuse beginners and practitioners alike , until understood that is , are meta compiler words scattered within the Kernel86.blk. But after L&P Forth life is never the same again.

No difficulty has been experienced in handling arithmetic and logic operators in postfix mode beyond the normal initialization stage. In the first 4 weeks step by step solutions to simple problems are deliberately undertaken to demonstrate how a word representing the final solution is made of simpler words. The necessity of documenting the interface between words thru a word's stack diagram is stressed. Students are advised to consult Leo's Brodie Thinking Forth.

Beginners face 4 types of difficulties. Clerical , Mechanical , Incidental and genuine Forth.

Clerical difficulties are due to the novelty of Forth. Inserting a space between words is usually forgotten. specially before and after colon , semicolon and dot-quote. Forgetting to terminate a definition with a semicolon . Not pairing IF .. THEN , IF ..ELSE ..THEN , DO .. LOOP ..etc and incorrectly nesting them . Also forgetting to use them only inside colon definitions.

Mechanical difficulties are in the use of the editor and the handling of files. The first time I introduced the traditional line editor I realized that students were having difficulties. They could not appreciate the overlap of the editor and Forth. A clear demarcation between edit mode and "Forth" was necessary. A screen editor was implemented. It is a must for beginners. But even then the screen editor has a learning curve.

The handling of files is strictly not Forth. Creating and opening of files. Moving blocks within and between files. Incidental difficulties are due to the necessity of learning the 8086 instruction set and its idiosyncrasies in order to write low level Forth words. This is repeated for the Zilog Super8 CPU.

Genuine Forth difficulties in ascending order were:

1. BEGIN .....WHILE....REPEAT. The flag for WHILE ?.
2. Interpretation and compilation.
3. Immediate words.
4. WORD . How come WORD usage is infix ? Its input is from input stream or stack ?
5. Deferred compilation.
6. CREATE .... DOES>

5 SUGGESTIONS

5.1 EDITOR

Introduce to the EDITOR a software switch which when activated will perform some preliminary error checking (with override facility) on a screen before storing it to disk In case of error an appropriate message is displayed . The instructor will save himself quite some effort in the first few weeks of the course until the students acquire a set of Forth habits.

The editor is to check that the following words are:

1. Preceded and followed by a blank:

   \ \S .(: ; "

2. Followed by a blank:) and "

3. Paired:

   DO ..... LOOP
   IF ..... ELSE .....THEN
   BEGIN .....AGAIN
   BEGIN .....UNTIL
   BEGIN ... WHILE .....REPEAT

5.2 FORTH

1. Software switch to filter out the input of control characters via keyboard.
2. Disable unused keyboard keys. e.g. Up arrow.
3. Provide Video and Sound words e.g.
   Pitch (S n --) generates sound of frequency n.
4. Word like EXPECT to accept only numbers as input.
5. Read and write ASCII and hex files at other than on 1 kilobyte boundary.
6. Help facility like VIEW but more in the nature of a tutorial and how to use.

5.3 COURSE

The topics of this course as outlined should be spread over 2 semesters with more time spent on hardware and on problem solving techniques. A specific set of labs and a project should be undertaken by the participants.

6 EVALUATION

It difficult to measure the proficiency of students as Forth programmers at the end of the course. Not enough effort was devoted to Forth programming techniques due to lack of time but instead students were advised to consult Leo's BRODIE " Thinking Forth" .

But consider the following : In one semester students learned a new language and assembled a forth computer kit using one of the many processors available. They generated a turnKey application . They metacompiled their own Forth program. They wrote short machine language routines for the Intel 8086 and the Zilog Super8 mixing it with high level Forth. They executed MS-DOS interrupt calls and gained an insight in the working of an operating system and a modern language.

The confidence and knowledge gained by such activity is considerable. It self motivates. That this has been achieved in 1 semester in an Electrical and not a Computing School without the traditional backup of prerequisite and corequisite subjects available at the undergraduate level reflects on the intrinsic merit of the Forth Language itself. Forth is an efficient and potent education tool. Reinforcement is ever present and instantaneous whether in the form of reward (ok) or punishment (? or crash). The final software/hardware product is quickly and easily realized leaving no time for boredom to set in or cause for disheartening. There is no constraint imposed by form. Creativity is not distracted or stifled it is liberated.

Some students have expressed their desire to attend an advanced follow up course. It was gratifying to find out that an organization having 1 of their personnel attend the course last semester followed it up by sending 4 more during daytime this year. Our School has officially adopted the subject which can now be offered anywhere in the State. The option is there to introduce it at a later time as an undergraduate subject . Plans to organize an intensive 2 day course on the NOVIX chip have been discussed at NSCT.

At the moment PASCAL is offered in stage 1. Following the lead set by Forth both FORTRAN and C are to be offered on demand starting next semester . BASIC is being phased out. Aware of the potential of Forth as an education tool and as a hardware tool I expect it to make further in roads in syllabus of early undergraduate stages.

7 ACKNOWLEDGMENT

It is no easy task to introduce to an institution conservative by nature with an established curriculum and set of subjects the notion of experimenting with new ,untried and unconventional ideas.. That I was encouraged to do so 2 years ago is a credit to staff who backed me up at NSCT. That decision is today fully justified by the advent of the NOVIX chip and the news of second sourcing by Harris semiconductors on the high end of the market and the availability of Forth for the Zilog Super8, the Intel 8031 and Motorola 68HC11 on the low end, by events like this Symposium and class attendance.

I would like to thank J.Haines,J.Goulding and B.Webster at NSCT for their unqualified support during the last 2 years , T.Rofe and P.Tregeagle of SYDNEY FIG for their help and NSCT Forth students for their help and enthusiasm which made the effort worthwhile, L&P for allowing us to share a masterpiece and C.Moore for creating Forth in software and hardware.

Computer Assisted Laboratories with ASYST.

Dr. A. Pugatschew
School of Applied Physics
South Australian Institute of Technology

Abstract

The intelligent and productive use of micro-computers in undergraduate Physics laboratories is an important opportunity to introduce a modern scientific tool and the assosciated techniques of data acquisition and analysis. The computer can be used in many ways --- equipment control, data analysis and assisting the student comprehension of physics principles. The successful integration of these multi-purpose experiments requires a new view of the laboratory aims and its importance. In fact, effort must be made not to produce a completely automatic experiment just because there is a need for some computer based practicals. In this paper we discuss the results of a programme commenced in 1985 that has the following objectives :

* Expand the experiments into the second and third year laboratories.
* Ensure productivity and efficiency with appropriate choice of language.
* Choose and design appropriate equipment.
* Design experiemts that can be shared with other tertiary institutions.
* Design practicals that permit some degree of student participation in both design and programming.

Introduction.

The important guidelines for this project were set after discussion with staff responsible for teaching laboratory development at S.A.I.T and the School of Physical Sciences, Flinders University. Many points were raised and it was clear that, although many students were computer literate, the disciplines of problem solving were inadequate. The lack of formal processes would restrict many of the aims.

The main points which would be adhered in this project are listed as follows.

* Introduction of data acquisition systems.

It is important to introduce some aspects of data acquisition systems (DAS) to first year students. A good set of simple experiments to illustrate the limits and strenths of computer DAS is essential to develop a foundation.

* DAS experiment design.

A prime requirement was that these experiments must be designed judiciously to demonstrate the benefits of DAS in comparison to traditional manual techniques. The computer would be seen as an indispensable tool in some areas where a simpler technique would be inadequate. The computer would not be viewed as a gimmick.

* Computer assisted learning of DAS

The computer should be used to illustrate the various acquisition and processing algorithms of signal analysis. This was particularly important for the S.A.I.T effort due to recent criticisms of the Physics degree where, it wqas felt, there was inadequate coverage.

* Student involvement should be planned at all levels.

It was important that students be permitted to participate in the hardware and software design of the experiment. This would illustrate the planning and programming steps.

* DAS language choice

It was evident from an early stage that a correct choice of programming language would be crucial to the success of these projects. FORTH was seen as a language that is designed for process and machine control and it was decided to concentrate on this environment.

ASYST, from Macmillan Publishing (now handled by Keithley Instruments) is a FORTH based language with four main modules - SYSTEM, ANALYSIS, ACQUISITION and GPIB. The extensive glosssary covers nearly every mathematical and acquisition task. Furthermore, the excellent documentation and tutorials make it easy for students to gain expertise.

We have developed three major experiemts with these aims in mind. The language has also been used by Civil Aviation students and electrical engineering students to perform some signal processing tasks.

FOURIER TRANSFORM SPECTROSCOPY.

The Fourier transform is undoubtedly one of the most important mathematical operations with applications in diverse areas such as music, signal processing , optics etc. This latter area is very important because the electronic principles of frequency response are similar to descriptions of optical system performance.

The spectrometer experiment is conducted in the the Third year Optics laboratory and consists of a modified commercial Michelson Interferometer (BECK) fitted with a motor drive to move one of the mirrors.

Students are asked to set up the instrument with a variety of input light sources and visually scan the fringe pattern when the mirror is moved. The fringe visibility is related to the coherence length of the source and the link is made with the line width via the fourier transform. The interferometer information presented in time or distance is converted to frequency or wavelength.

The experiment is planned by defining ASYST functions which perform the operation on any input array and , in order to test these words , a mathematically generated fringe pattern is analyzed. The number of fringes is progressively truncated and the resultant changes in the line width are presented. The results of this first step which introduces the students to the FORTH philosophy are shown in figs. 1. 2. and 3.

In the experiemntal situation the optical fringe pattern is focussed onto a photo-diode and sampled with a DT2801A acquisition board fitted in an IBM PC . The fringe pattern analysis and graphics presentation is performed with ASYST.

The fringe patterns and the resultant line features for several light sources are shown in figs. 4,5 and 6 . The students were able to calibrate the wavelength scale of the tranformed data an produce a calibration graph (fig. 7).

This experiment was performed in 10 hours and the students have had no prior experience with ASYST or the DAS system.

TORSIONAL PENDULUM.

Forced harmonic motion and resonance are concepts that are very important in many technologies (building science, electronics etc). The torsional pendulum is a simple experiment to demonstrate the variation in amplitude of a driven system as a function of the driver frequency. Unfortunately, the measurement of several data sets for various degrees of damping is difficult to perform in the time allocated in the first year laboratory.

A DAS version was constructed where two servo potentiometers sample the angular displacement of the upper and lower rotating discs. An 8 bit converter board is used to illustrate the poor accuracy at the low and high frequencies where the pendulum amplitude is low. The signal is smoothed and averaged in order to determine the phase and amplitude.

Results obtained are shown in fig 8.

FALLING BALL experiment.

The motion of freely falling objects and motion down an inclined plane are common, simple mechanics experiments. Data is obtained by interrupting light beam - sensor arrays positioned down the incline and distance versus time and time squared graphs can be produced. However, the accuracy can be poor unless good timing instrumentation is used. The first year experiment uses a storage oscilloscope to display a the sensor outputs. It was decided to use ASYST to analyse these outputs and produce time versus position data. The system was set up in approximately half a day and allowed measurements as a function of the angle incline to be easily achieved. Several manual measurements and analyses are performed at the same time.

Sample outputs for the raw data and the final graphs are shown in figs. 9,10,11.

Summary.

Selection of physics experiments that use a microcomputer system in an intelligent fashion are very useful in an under-graduate physics course. The time saving aspect permits more results to be obtained and more physics knowledge to be presented. This programme relies heavily on the use of an efficient language -- ASYST.

Acknowledgements.

Assistance from the Tertiary Education Authority of South Australia and Logiteck is gratefully acknowledged.

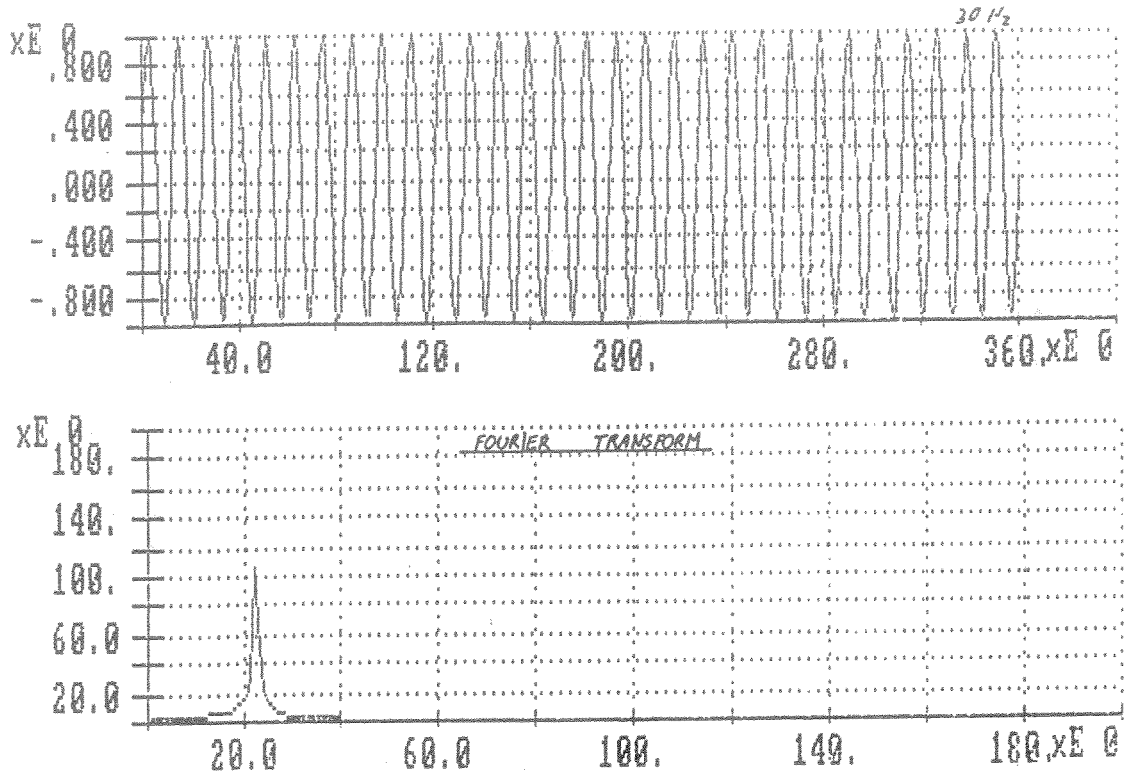


Figure 1

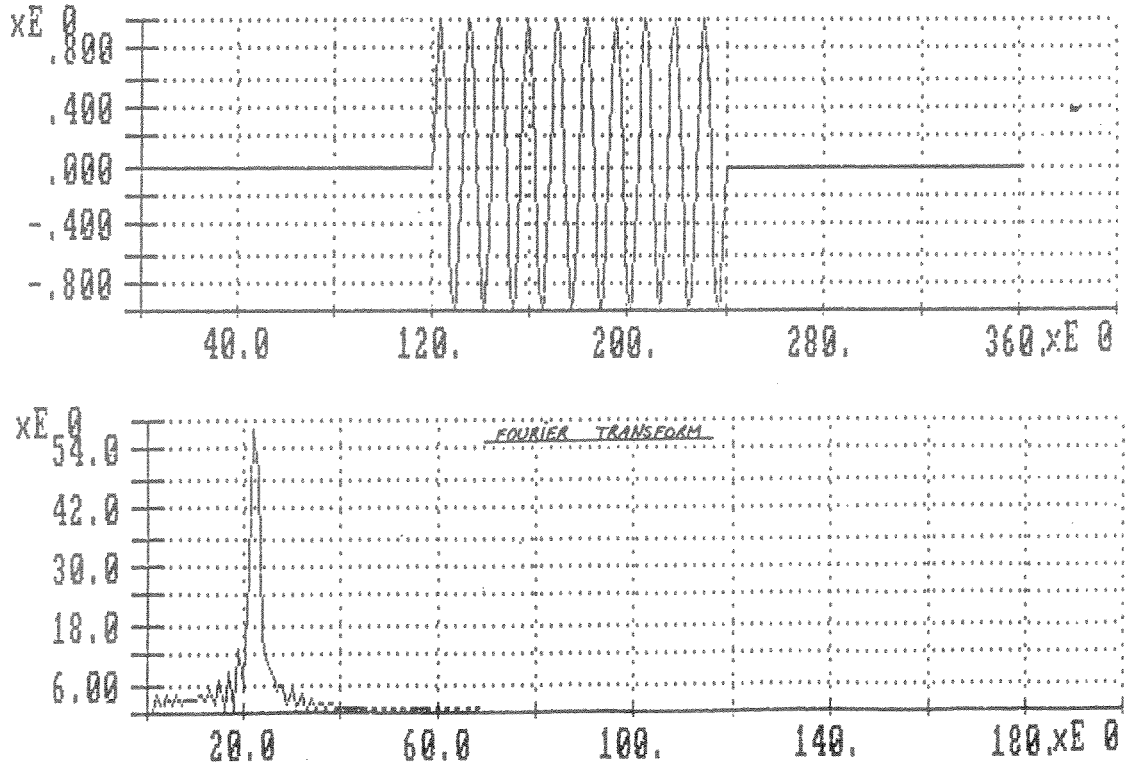


Figure 2

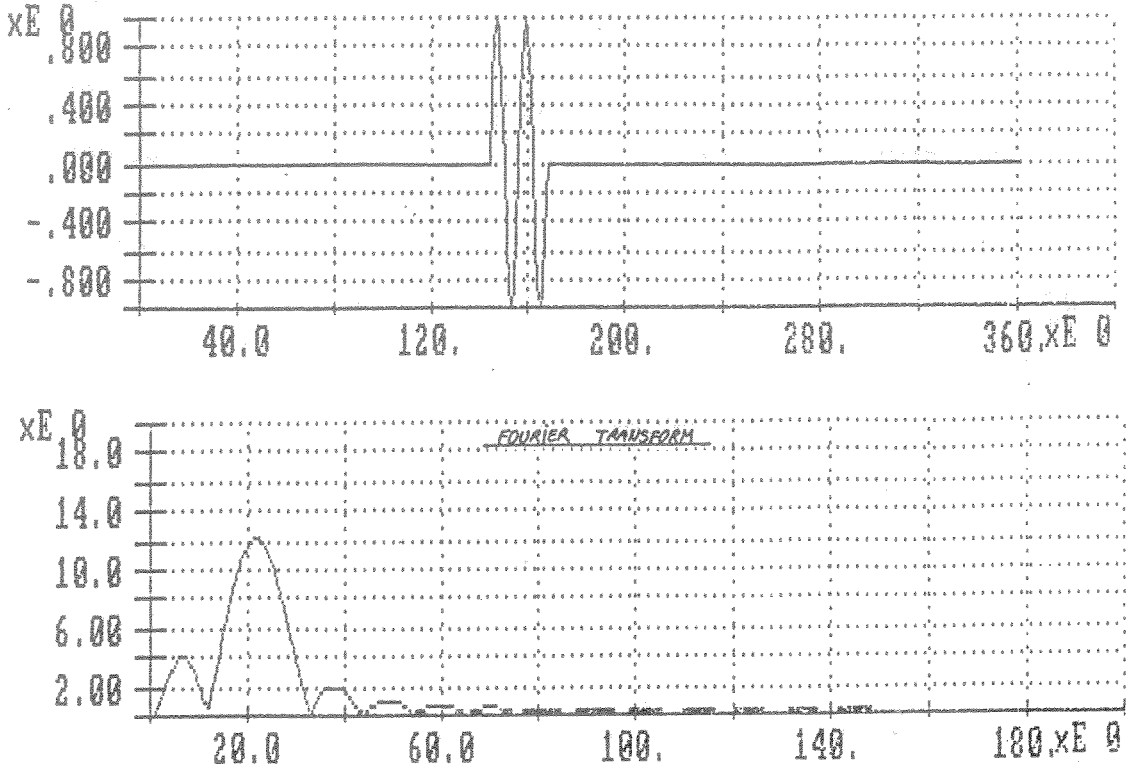

xE 0
.800
.400
.000
-.400
-.800
40.0
120.
200.
280.
360.xE 0
xE 0
FOURIER TRANSFORM
18.0
14.0
10.0
6.00
2.00
20.0
60.0
100.
140.
180.xE 0

Figure 3

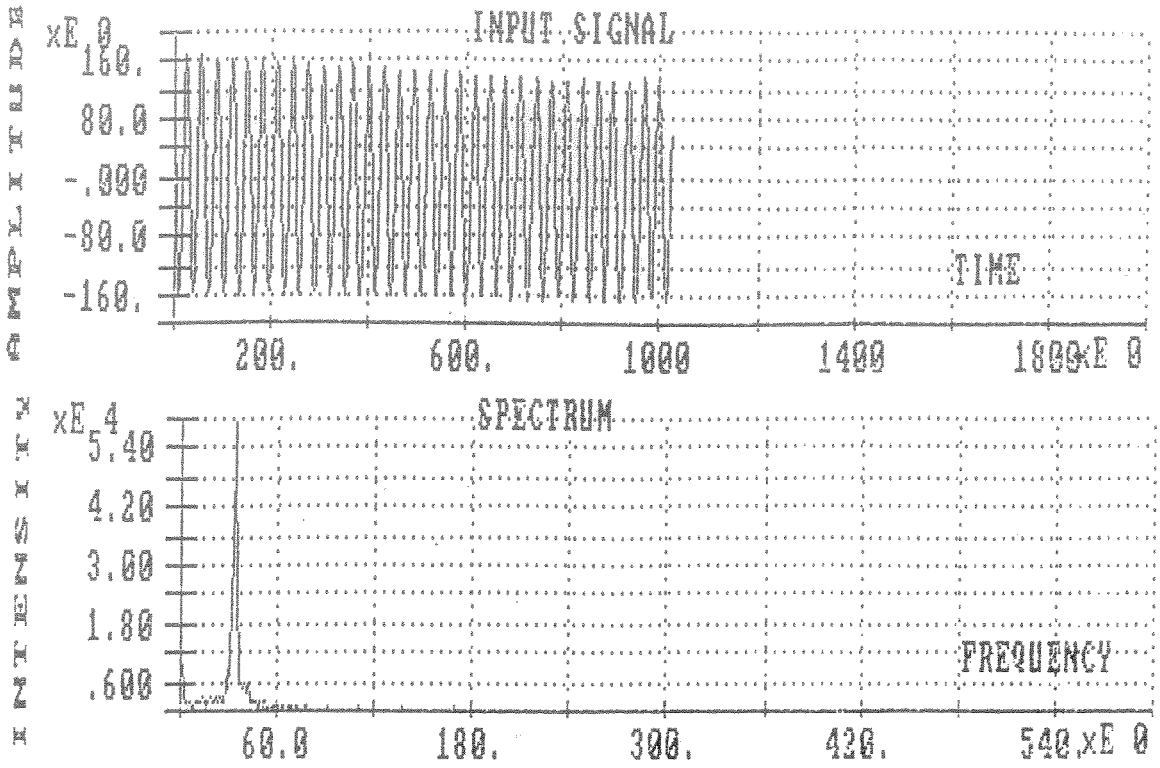


Figure 4. He-Ne Fringes and Transform

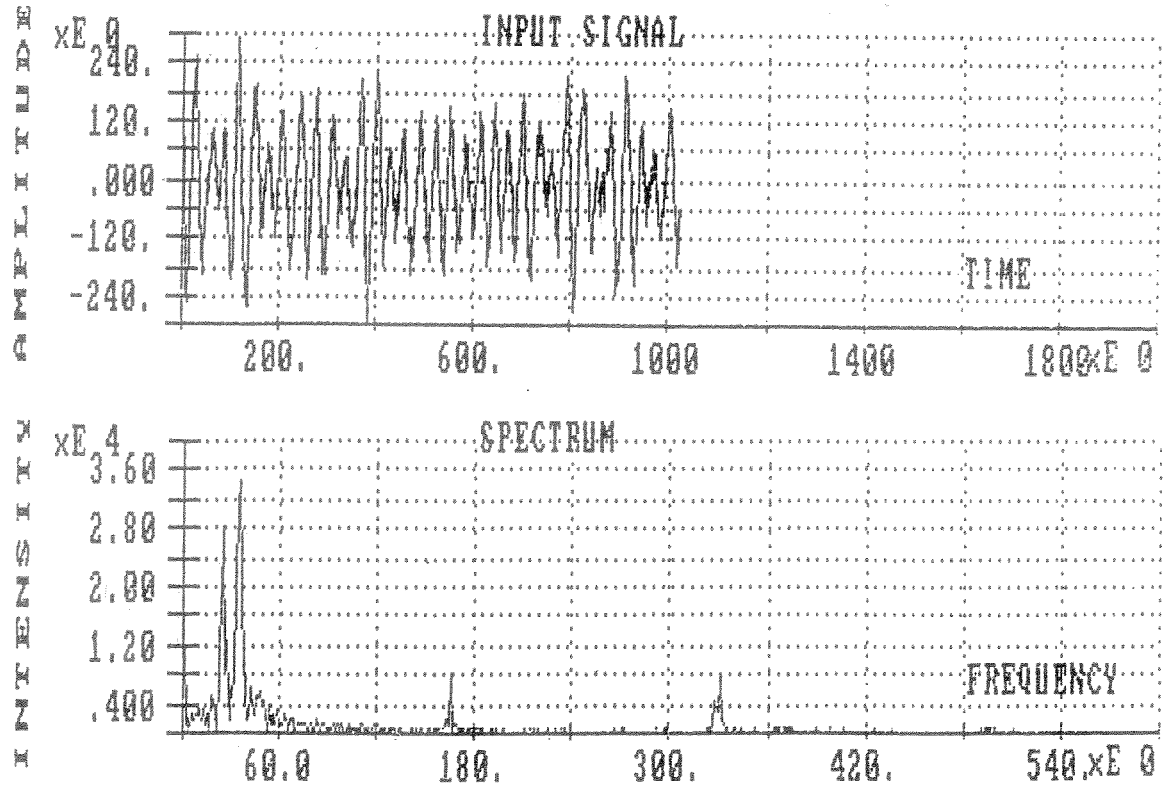


Figure 5. Sodium Lamp Fringes and Transform

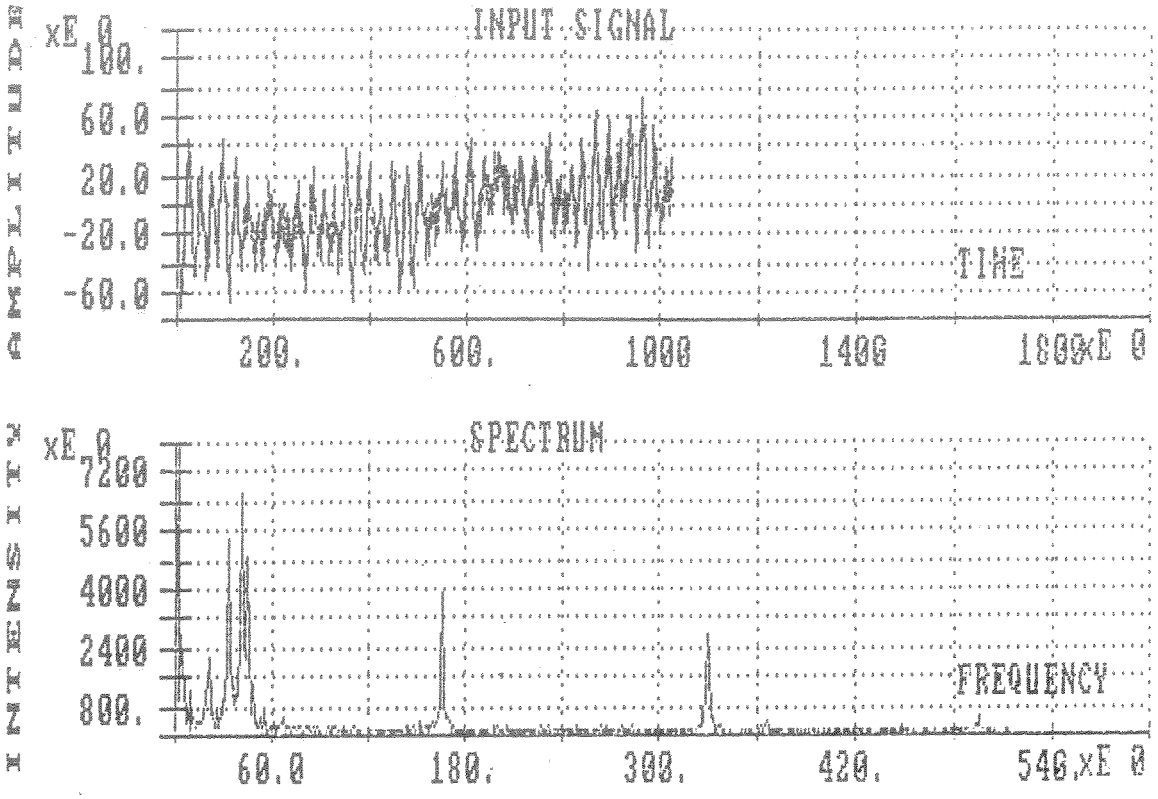


Figure 6. Cadmium Lamp Fringes and Spectrum

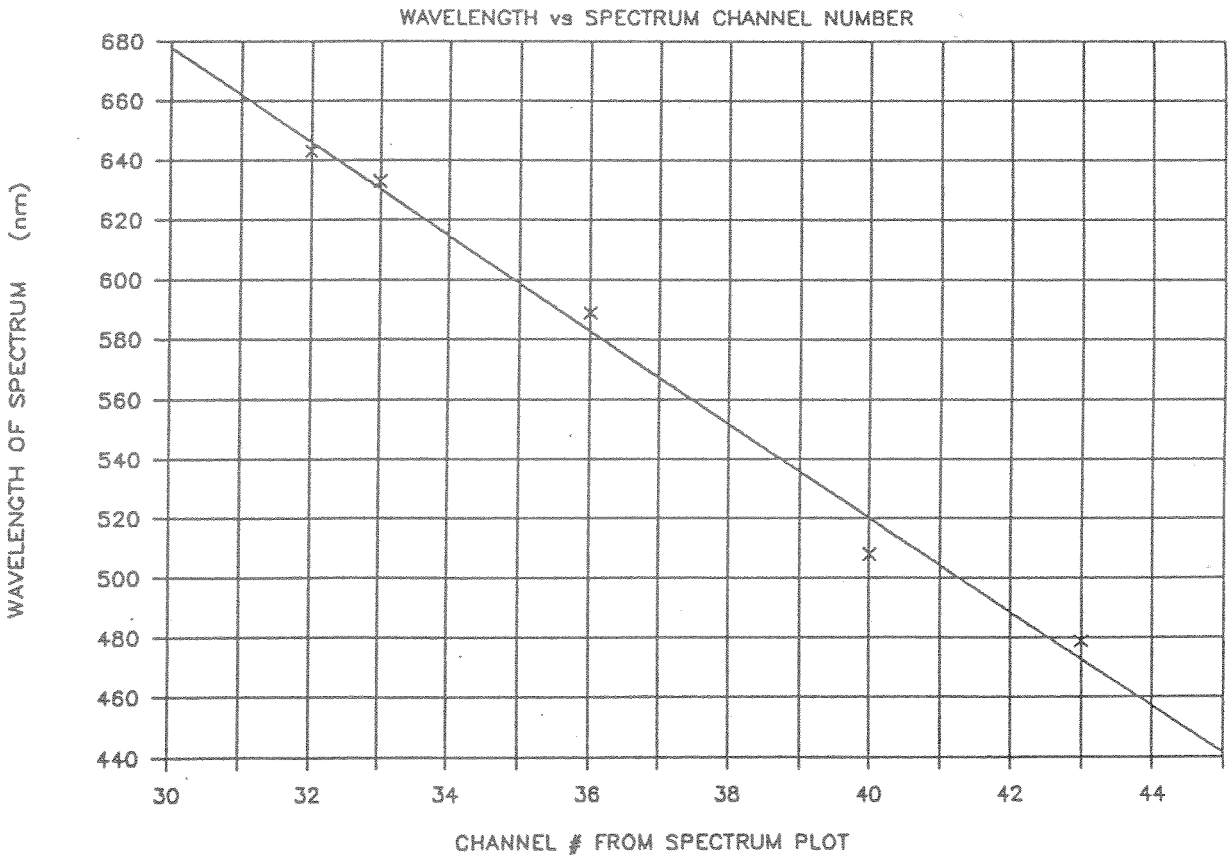


Figure 7. Calibration Curve

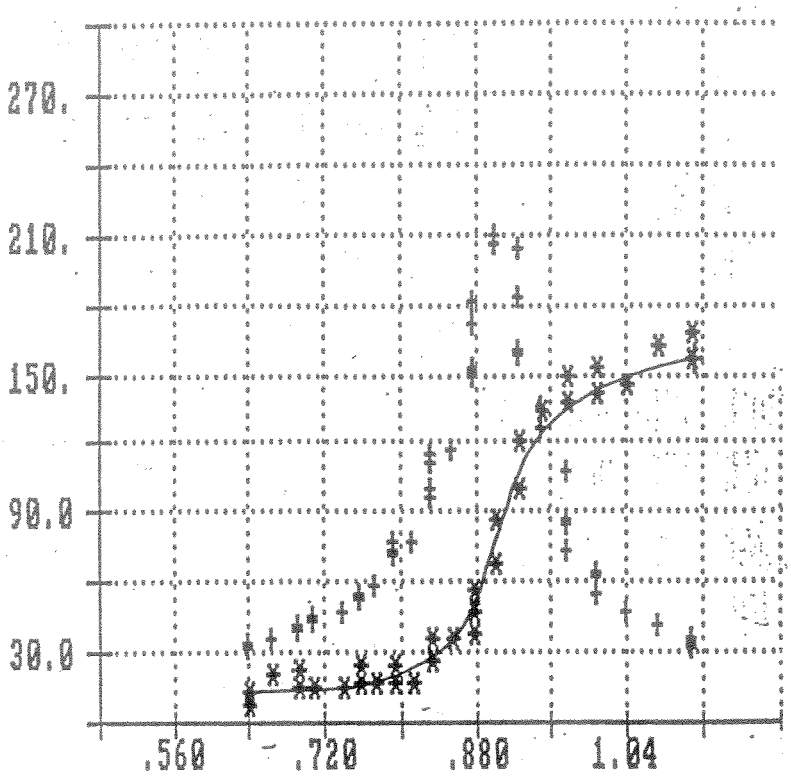


Figure 8. Plot of Amplitude and Phase as a Function of Frequency

* amplitude

\+ phase

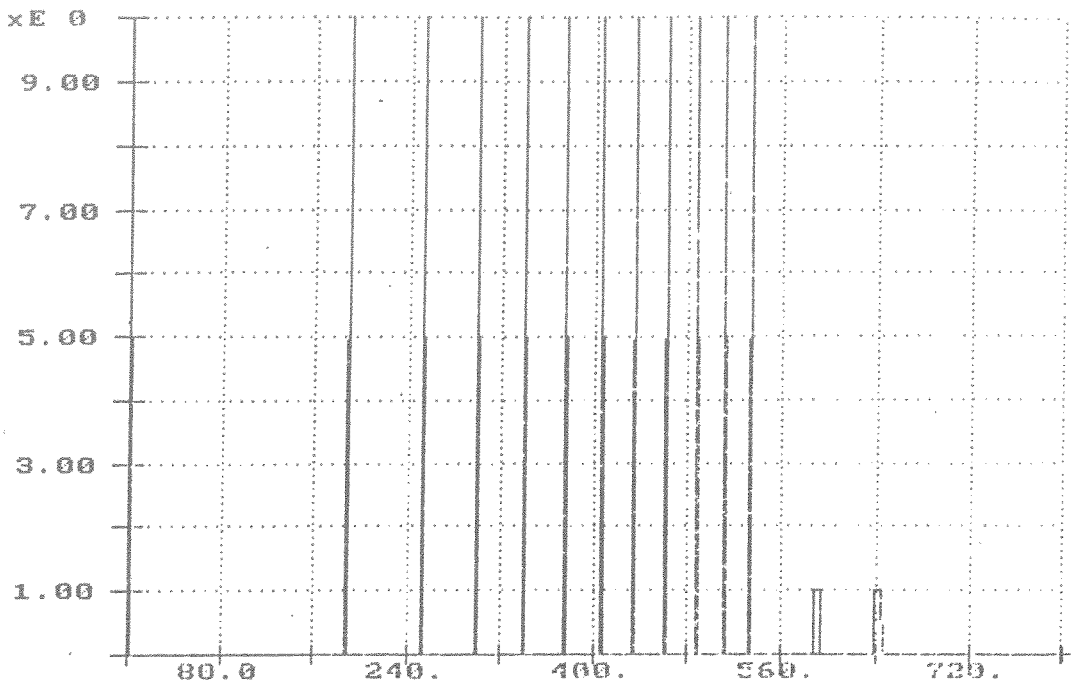


Figure 9. Timing Marks versus Time in milliseconds

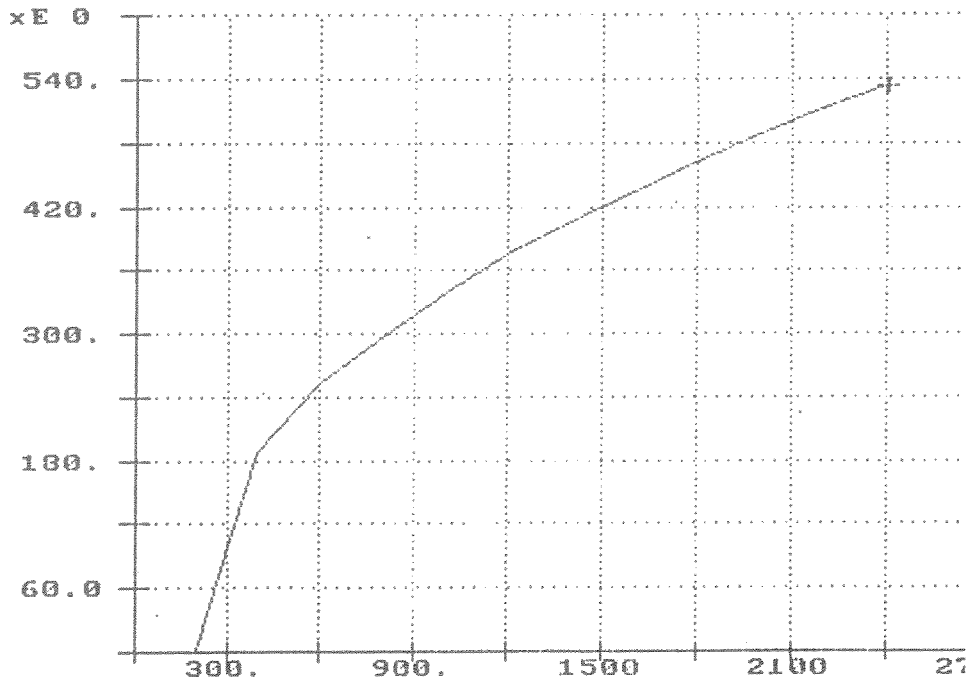


Figure 10. Time (msecs) versus Distance (mm)

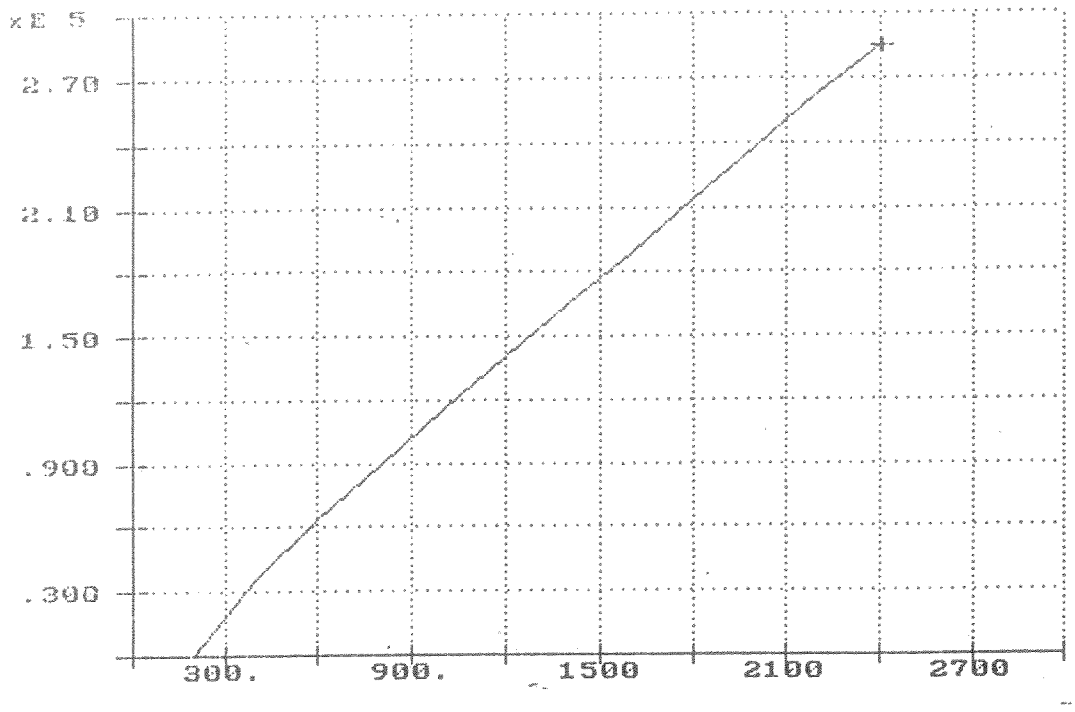


Figure 11. Time (squared) versus Distance (mm)

Simulations for CAL in Forth

I.P. Walsh - Scientific Officer
D.E. Bailey - Senior Lecturer

School of Science and Technology, Nepean College of Advanced Education.

The Physics Literacy Project was initiated in 1984 with funding supplied by the Commonwealth Tertiary Education Commission under the Participation in Equity Program. The project had as its aim the production of computer assisted learning programs to assist students with little or no background in Physics to gain necessary concepts for success in tertiary Physics courses. The project was initiated using Apple Macintosh computers and the programming language Forth. Four years down the track, a number of packages have been generated that emphasise screen production of simulations of Physics events. The emphasis of this paper is not in the origins of the project and the final product obtained, nor of the uses to which it will be put, but on the lessons learnt in the process of producing the graphic sequences and on the language used, MacForth. The paper is written in the first person because it mainly concerns Ian Walsh's experiences as the most recent programmer with this project.

I joined the Physics Literacy Project at the beginning of 1987 after having spent sixteen years teaching Mathematics at high schools in N.S.W. During that time I took more than a passing interest in computers. In the days just before personal computers were introduced I learnt to program the Sharp programmable calculator. Before I left teaching I saw the need for a program to record results at swimming and athletics carnivals. As a result of this I then wrote a program for the Apple II computer in Basic and used the Microsoft Applesoft Basic Compiler to compile it.

For years I had resisted learning how to use a Hewlett Packard calculator but now I was to learn all about reverse Polish notation and make it part of my automatic thought processes. As I have said, I had a very good working knowledge of Basic, and I had also been exposed to Algol, Fortran, Cobol and Pascal at university and while teaching, but I had only heard of Forth through my brother who wanted to get his hands on a copy of Forth for his Macintosh. My knowledge of Macintoshes was to know my way around MacWrite and MacPaint and that was all. (I used a Macintosh to produce the documentation which accompanies the carnival recording program.) I was given a Mac and a copy of MacForth Levels 1, 2 and 3 and a number of books and was set to the task of learning to program in Forth on the Apple Macintosh. Learning Forth was a reasonably simple matter. I had a working knowledge of Forth within a few weeks and had produced a simple music program in that time.

The biggest break I had was coming in on the project just after Creative Solutions released a completely new version of MacForth - MacForth Plus. This version offered a vastly superior working environment. MacForth Plus has a very good text editor using ordinary text files and is not restricted to the traditional screen of 1024 characters. No more "blocks" files! Forth files are now much smaller than they used to be and there is no wasted white space. MacForth Plus includes command line editing, multi-tasking and turnkey capability and it runs much faster than the earlier version. MacForth comes with a basic Kernel, the latest is 3.53, and quite a number of extension files containing the source code for such things as the editor, controls, advanced graphics, printing, multi-tasking, floating point and assembler. A number of sample programs are included to enhance the manual and further assist the user.

My main hurdle was to try to achieve smooth motion, since all of the programs for the Project would involve graphics sequences. I had great trouble finding anything in published form which was of use to me in my quest for motion techniques. I had to resort to trial and error methods in my pursuit of realistic looking motion. Some of my early programs used the Macintosh's powerful inbuilt region procedures to draw and undraw the parts of the moving object which change. Later, after I had looked at the 3D package from Creative Solutions, I put together my own file of bitmap words which I use now in virtually every program I produce. I achieve smooth animation by copying a small offscreen bitmap to the screen in synchronisation with the system's inbuilt 1/60 second clock.

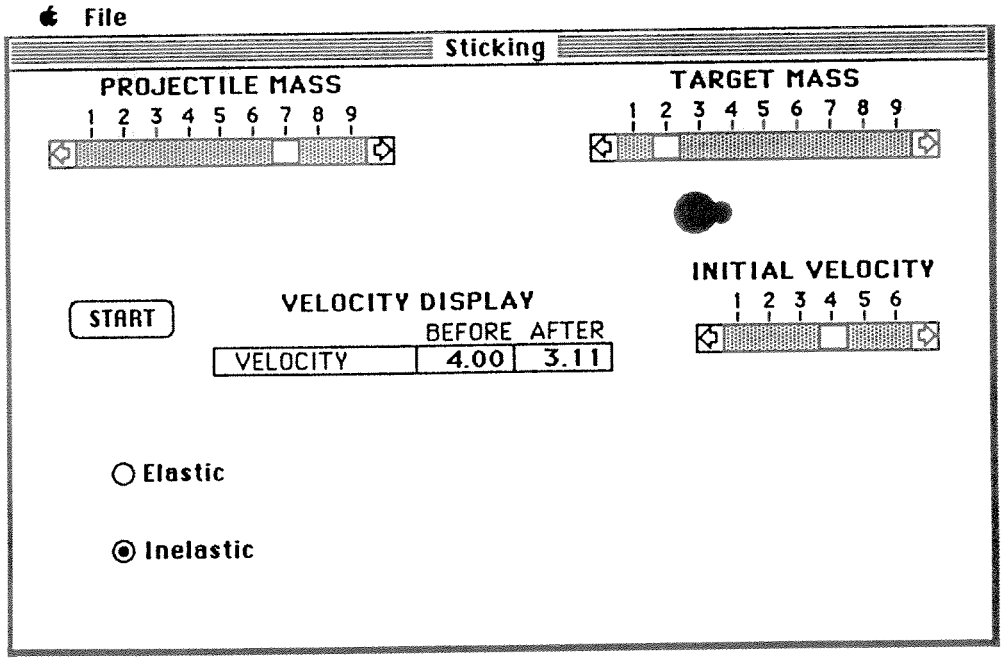
In the twelve months on the Project (Bailey 1987) I produced about sixteen finished programs. I also completed a music program in Forth for a music lecturer at Nepean C.A.E. and another series of three programs for the University of Technology. My best time for a program was ten hours for one of the DC Circuit programs with an average of fifty to sixty hours for most of the programs. I estimate it took me about four weeks to learn Forth and about three to four months to learn about programming the Macintosh. It is true that, when using the Macintosh, you can virtually throw away the manuals, but when you are programming it you need a stack of technical manuals. MacForth comes with a huge dictionary of predefined words. This enables a programmer to access the large number of toolbox routines and functions built in to the ROM on the Mac. MacForth has over two hundred toolbox traps predefined for the programmer. The rest of the toolbox routines can be accessed reasonably easily from MacForth by using some predefined words for setting up the needed stack frames. If the appropriate stack frame has not been predefined then the frame can be set up using some predefined micro-assembler words which allow you to manipulate certain registers without compiling the full assembler package.

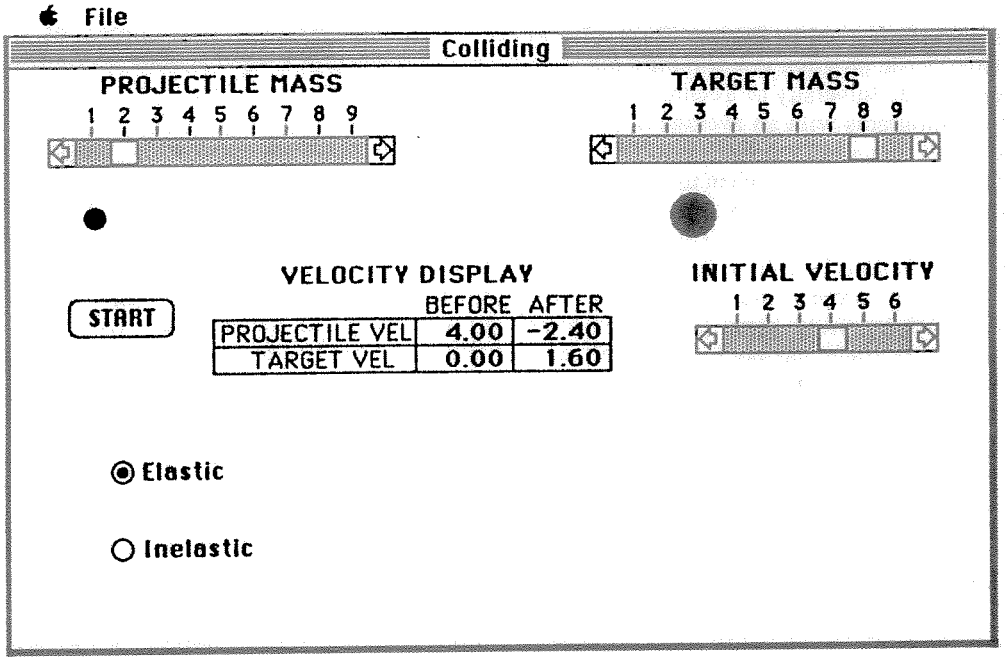
Having come from a Basic background with no commercial experience with C , Cobol or Pascal it is hard for me to comment on the productivity of Forth over other languages. For me the best feature of Forth is to be able to write a word, put it in the dictionary, that is 'compile it', and then test it straightaway. It enables me to be extremely confident that I will not have to spend hours chasing bugs around if the testing is done incrementally.

An advantage of MacForth is that it comes with an in-line assembler. When a program is running, but you are unhappy with the speed of some part of it, some of the Forth words can be recoded in assembler very easily. Creative Solutions have said that most of the important Forth words in the kernel have already been coded in assembler and I have only used assembler in two programs, **Fields** and **Hooke's Law**. In **Fields** I achieved a big increase in speed but in **Hooke's Law** it was probably not worth the effort.

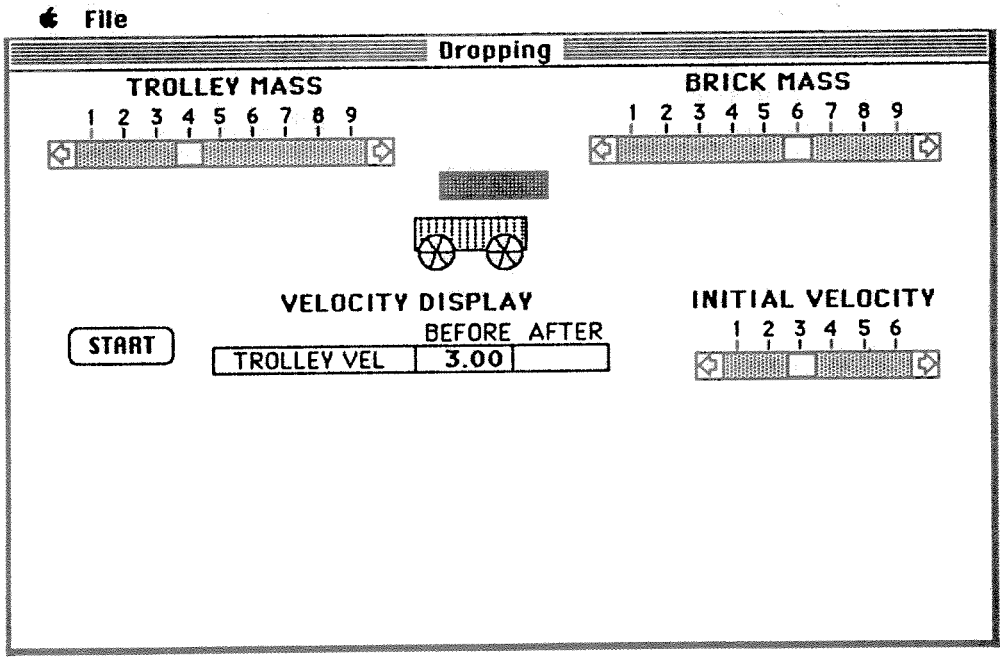
A disadvantage to using MacForth is that it is supported from the U.S.A. but if you are prepared to use the telephone at five or six in the morning then they have a telephone hotline available for an hour each day. Support is mainly offered on CSI Forth Forum on Compuserve, an electronic bulletin board. I would suggest that you do not join the National MacForth Users' Group. I sent my money in March, 1987 and, to date (April, 1988), have received absolutely no response. This group is independent of Creative Solutions.

I have included here a number of screen dumps showing most of the programs which have been developed using Forth. Also included is a very quick outline of the method used to get the end result. A number of monographs are available which include the Forth code and highlights of the methods used in more detail.

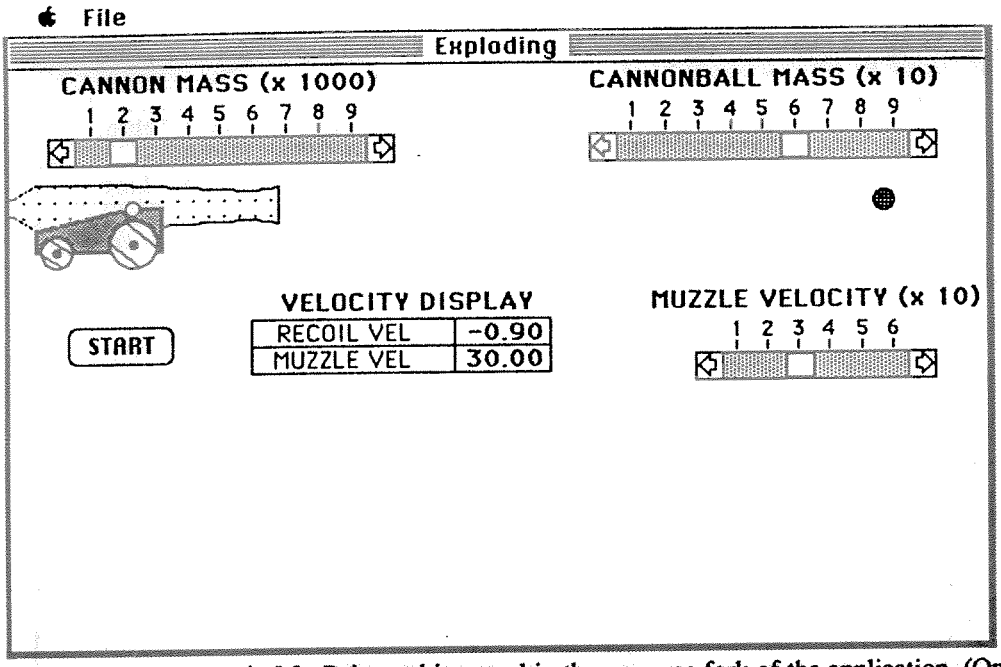




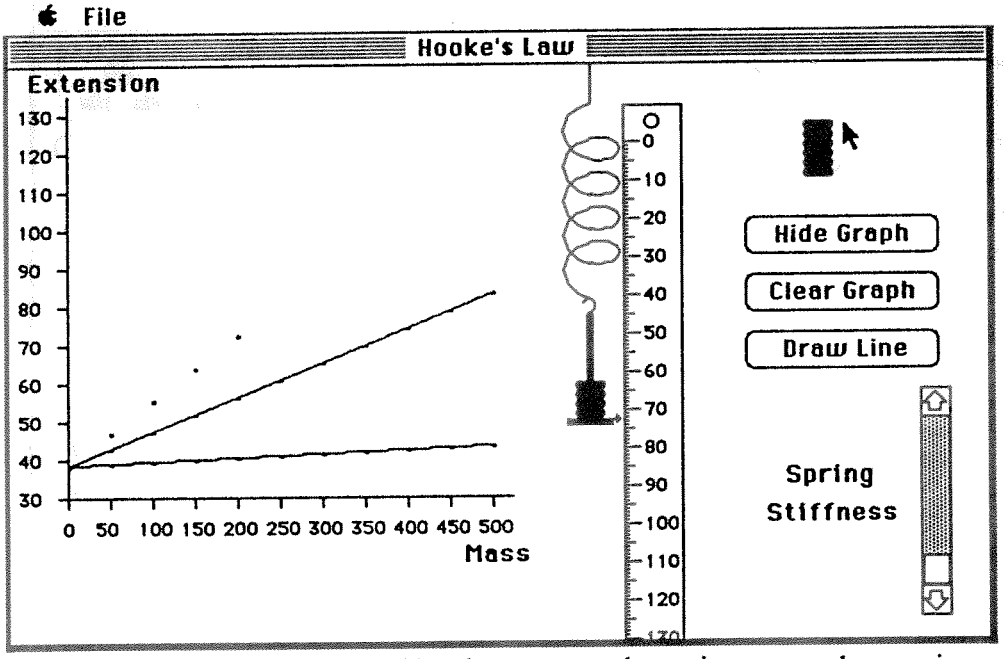
Originally this motion was achieved using the toolbox region procedures to calculate the new areas to draw and undraw, as it would be entirely unsatisfactory to draw and undraw the entire masses because of the flicker obtained. The effect obtained using the region method was reasonably satisfactory, but later on it was changed to a bitmap method which was noticeably better.



Again a bitmap method is used here. The trolley and brick are actually stored in memory in the same bitmap but clip regions are used, before dropping, to draw each of them separately. The trolley bitmap has a blank area six pixels wide behind the trolley. This is used to erase the previous trolley on the screen since the maximum distance that the trolley is moved at once is six pixels.

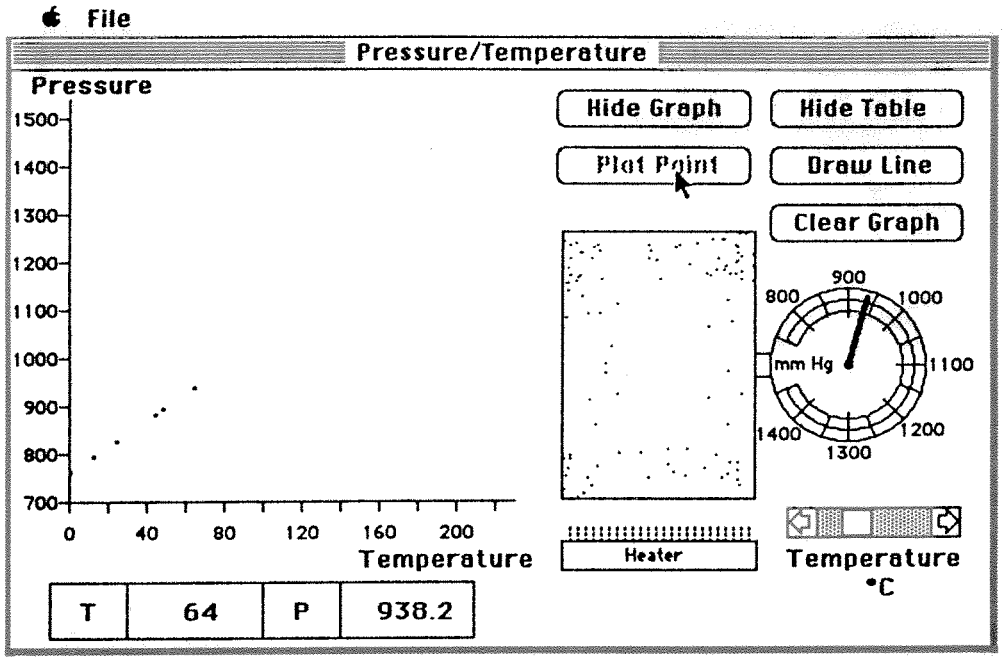


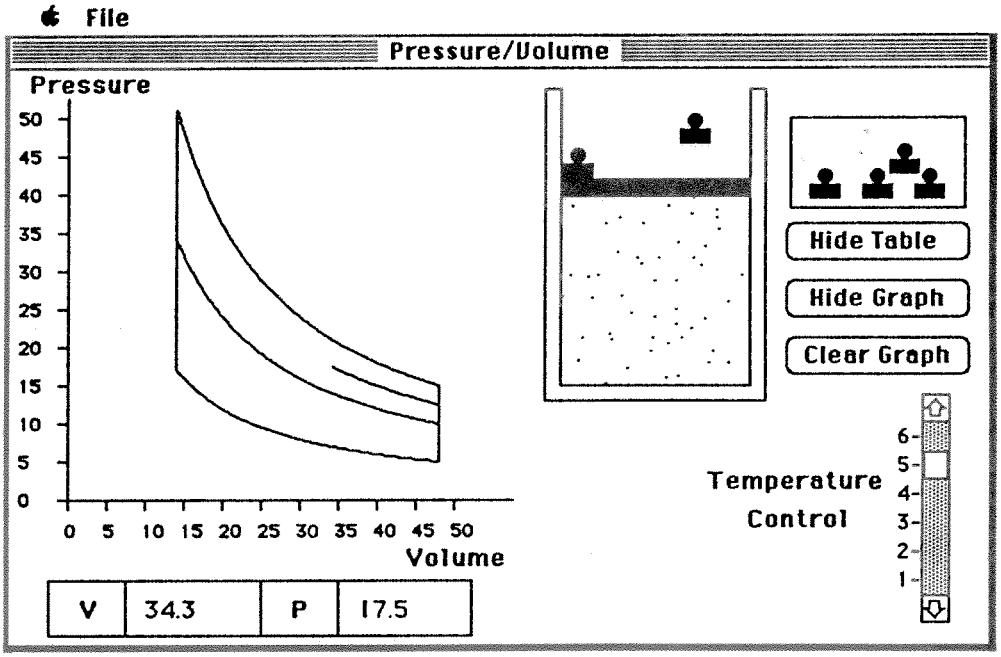
The cannon has been drawn in MacPaint and is stored in the resource fork of the application. (On the Macintosh an application consists of two parts, the code part and the resource part.) It is copied to an offscreen bitmap at the beginning of the program. It was found, that to actually draw the spokes each time was too slow, and so 66 small bitmaps were set up with the spokes in different positions. There are 36 different positions for the spokes of the big wheel giving an angle of 5° between each position of the spokes, since a rotation of 180° will bring the spokes back to their initial position. Similarly there are 30 different positions for the small wheel giving an angle of 6° between each position of the spokes.



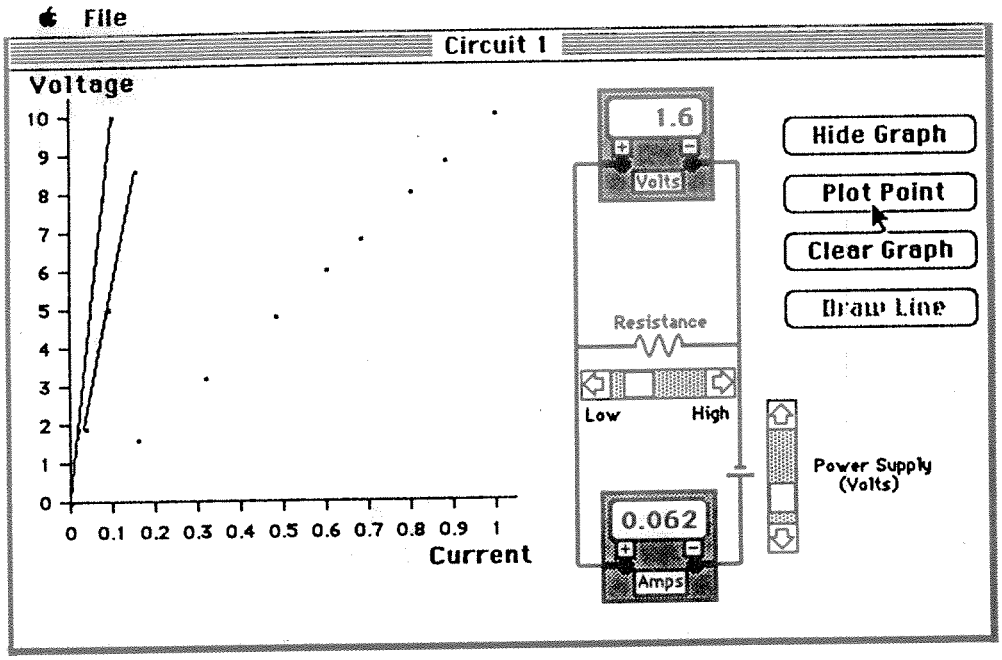
When a mouse.down event is detected in either the storage or the carrier areas, and a mass is present, the arrow cursor is changed to a mass cursor and a mass is undrawn. Each of the masses are separate regions. The carrier and spring are drawn to an offscreen bitmap before being copied to the screen.

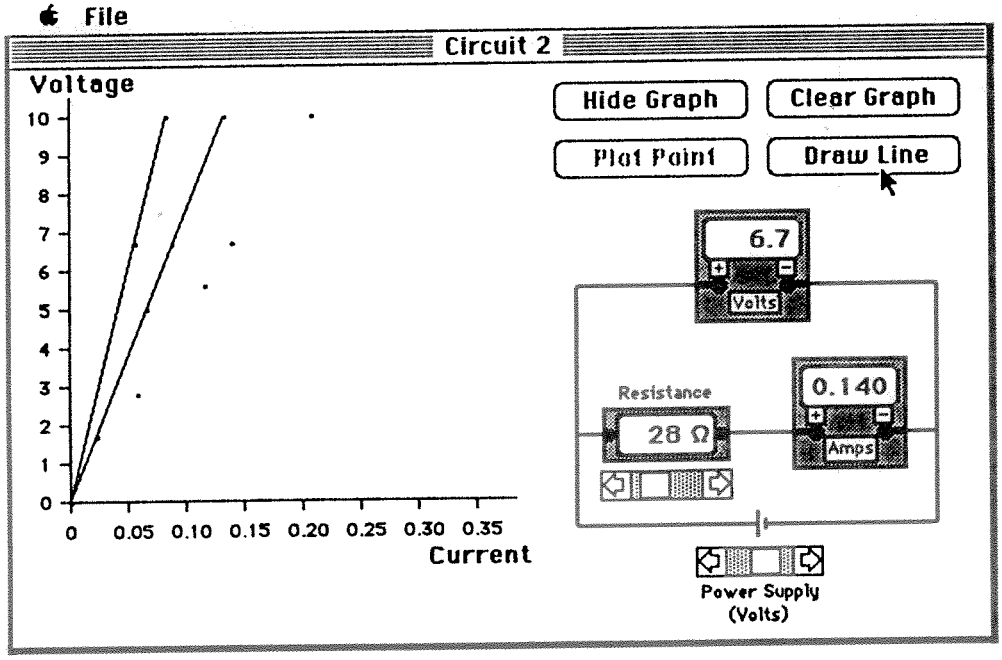
Originally the entire spring was drawn each time the spring was extended or compressed. This took about 21 tickcounts (21/60 second) to draw the spring. It was then changed so that only one and a half coils need be actually drawn and then the whole coil was copied four times to complete the entire spring. This procedure reduced the time to draw the spring to 8 tickcounts. It was then decided to code this up in assembler and this further reduced the time to 5 tickcounts. Whether it was worth the effort to code it in assembler is questionable.





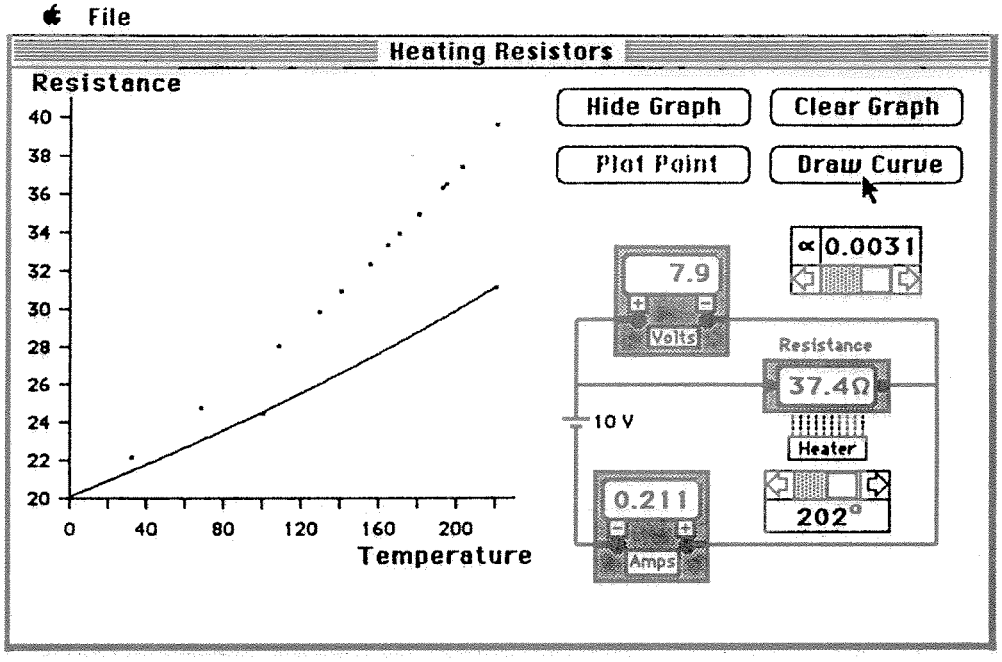
In **Pressure/Volume** the masses are added to the piston region when they are placed on the pressure vessel, making only one region to move. This region is moved by drawing and undrawing the essential parts only. The drawing of the gas molecules has been set up as a background task. The speed of these molecules varies with the temperature. In **Pressure/Temperature** the flickering flame has also been set up as a background task. The flame consists of three bitmaps which are alternately copied to the screen. The height of the bitmaps, which is determined by the temperature, is controlled by using one of twenty clip regions. Floating point arithmetic is used when drawing the graph in **Pressure/Volume**.



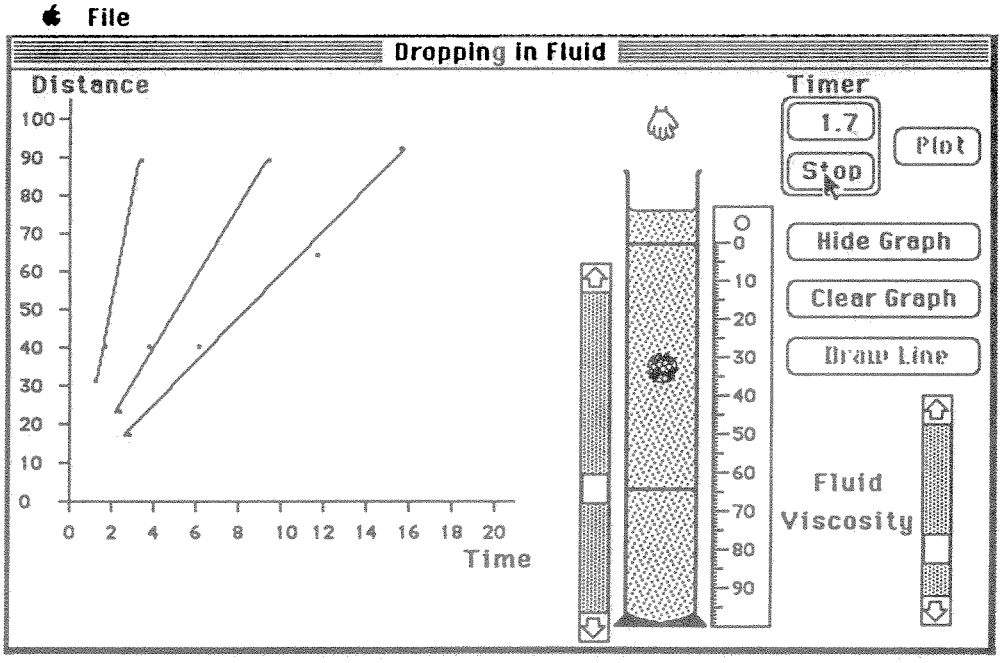


The circuit diagrams were drawn in MacPaint, stored in the resource fork of the application and copied to a bitmap at start up.

The graph is kept in an offscreen bitmap so that screen updates can be made of this area easily. This avoids having to "remember" what was drawn on the graph and to redraw the dots and lines each time. A bitmap of the blank graph is used when clearing the graph. This is done for speed reasons since memory is not usually a consideration nowadays.



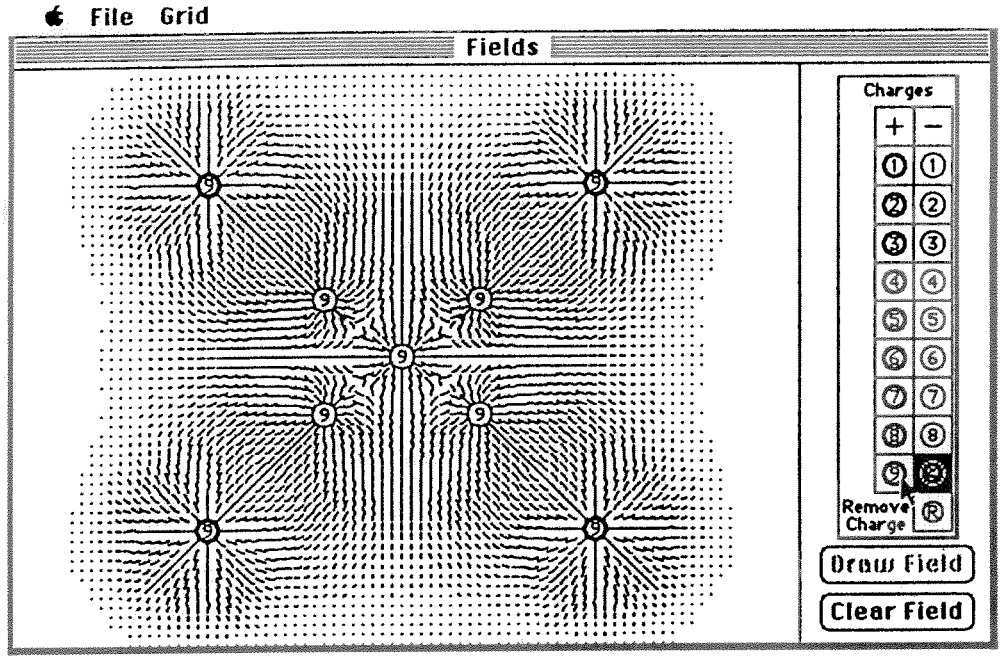
Once again the diagram was drawn in MacPaint and is stored in the resource fork of the application and the flame is a background task. Floating point arithmetic is used when drawing the graph.



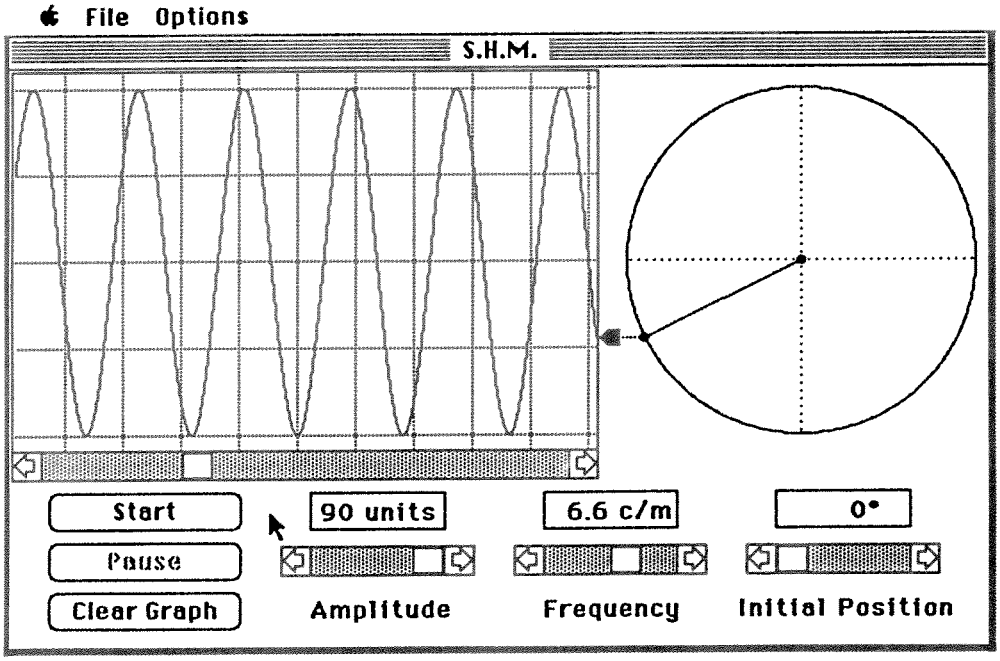
During dropping there are three separate stages. Firstly when the hand and ball move down together and then the hand moves up while the ball continues down. Secondly, the hand has stopped and the ball continues down until it hits the "fluid". Thirdly, the ball continues on until it hits the bottom of the cylinder or a key has been pressed. The quickest way to achieve the motion in this case was to use the **srcxor** mode to draw the ball, and then "undraw" it by drawing the ball again using the same mode.

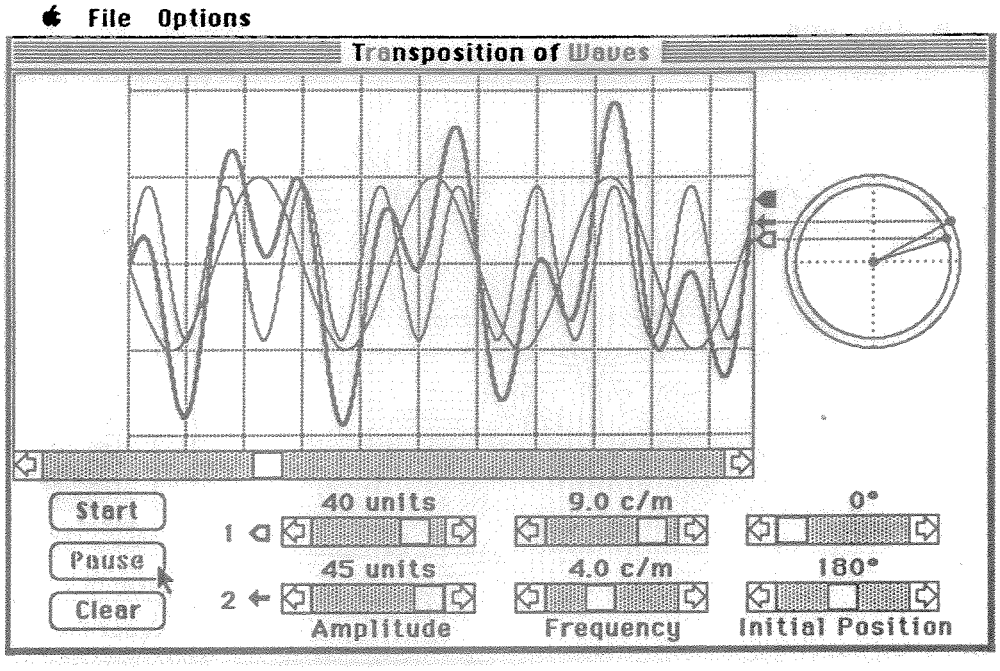
The ball, hand and cylinder were drawn in MacPaint and read in as resources.

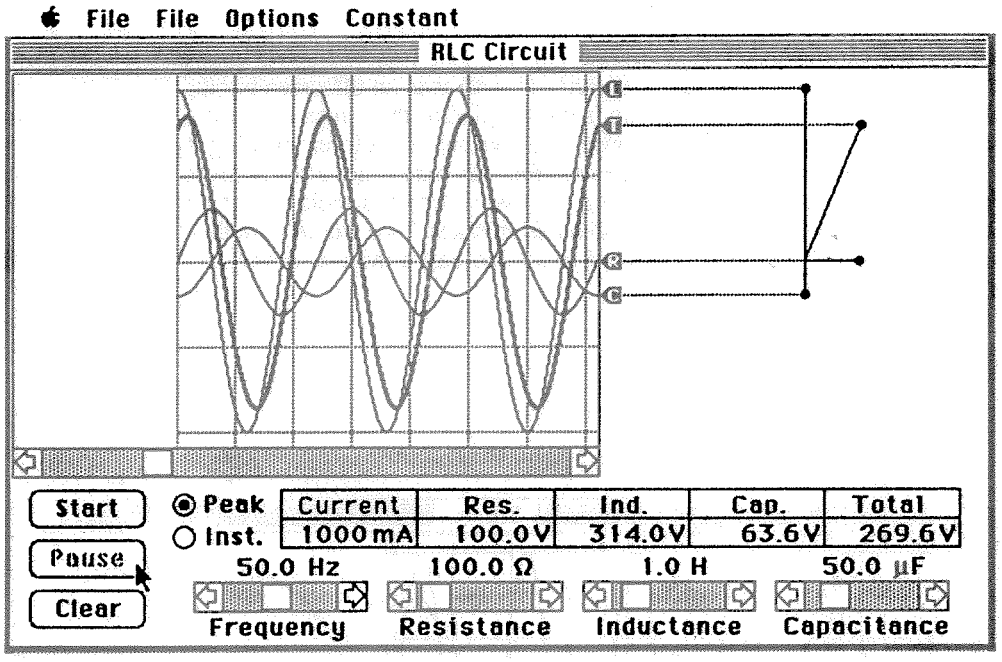
The timer has been set up as a background task.



The charges are drawn onto the screen directly as well as to an offscreen bitmap. The field is drawn to the offscreen bitmap, part of this bitmap is then copied to the screen after the charges in this area are redrawn. The field force vectors are stored in arrays and, therefore, updating the screen is very quick when another charge is added and a large number of charges are already present.When this program was originally written the screen drawing for twelve charges took almost three minutes. This was reduced with some optimisation to two and a half minutes. Two words were then coded up in assembler and the drawing time was further reduced to around half a minute. A very acceptable time and well worth the time spent learning assembler!







The graph in each case is maintained as a very large (about 1400 by 200) offscreen bitmap. This allows for a quick review of the graphs. The drawing is done offscreen and then the visible section of the graph is copied to the screen. The generating diagram is also drawn to an offscreen bitmap and copied back to the screen, this allows flicker free motion.

At present there are not many people using MacForth in Australia that we know of. If you would like assistance or can offer assistance to users, please let us know (047-360428) and we could form an informal mutual help group.

References

Bailey D.E., Logan P.F., Walker P.J., Walsh I.P.
The Physics Literacy Project- 1987, CALITE Conference 1987," Using Computers Intelligently in Tertiary Education", Barrett J. and Hedberg J. Eds, 1987.

MacForth Plus
Creative Solutions Inc., 4701 Randolph Road, Suite 12, Rockville, MD 20852 U.S.A.
(Telephone 301-984-0262)