Int. J. Engng Ed. Vol. 11, No. 1, pp. 58–66, 1995
Printed in Great Britain.

0742-0269/91 $3.00+0.00

Development of an Authoring System for Engineering Education for PC-DOS Applications*

P. P. GANAPATHI SUBRAMANIAM
S. S. GOKHALE
Department of Aerospace Engineering, IIT, Madras, India 600 036

The main aim of the current work was to develop an authoring system shell suitable for teaching courses in engineering disciplines. Such a system is designed to work for IBM-PC and compatible machines with MS-DOS. The code was generated using C++ and is best suited for VGA color monitors even though it is suitable for other types of monitors. There are no other hardware and software requirements. By incorporating an interpreter or program-like structures, it was found that one can construct an authoring system with modest capabilities, with a fast response and which does not require a large storage space. Such an authoring system is adequate for developing learning material for classroom applications. The incorporation of hypertext and hyperpage facilities enables the system to be applied to materials developed in a non-linear mode or for exploratory learning.

INTRODUCTION

THE PHENOMENAL growth in usage of personal computers in all walks of life in the past decade has opened new vistas for computer applications in education. Today computers provide opportunities for distance learning, they can partially make up for a lack of training facilities in institutions where there is a shortage of qualified teachers, as well as provide individualized instruction for learners with certain disabilities. PCs can also provide options for teachers to write their own computer programs to explain isolated topics. These programs, however, do not cover complete courses or extensive subject matter. For this type of work, one needs software to assist in preparing instructional packages. The purpose of computer-aided instruction (CAI)/computer-aided learning (CAL) packages is not merely to reproduce textbook material electronically but rather to prepare material that a teacher would present in a class for the student for independent learning.

Earlier efforts in the development of engineering education software were in terms of exploring the new medium of computers to enhance an existing approach to teaching and not to replace it. The main objective was to reach the learner more effectively and efficiently. Organization of such projects was by technical topics and not by the complete subject being taught. The end user of such a system was believed to be not a researcher attempting to verify a theory, but a person interested in learning about a particular topic. Project Athena [1] at Aerospace Engineering at MIT as well as Macavity [2] from MIT's Civil Engineering Department were programs or groups of programs to fulfil this objective. The former covered subject matter in the area of fluid mechanics, thermodynamics, propulsion and control theory, whereas the latter dealt with applications of static equilibrium and elementary strength of materials to determinate beams. The main characteristic of both projects was the use of high-end graphic workstations and modular developments of a number of programs. These programs were written by subject experts and were predominantly in FORTRAN and C with suitable graphical interfaces. However, as suggested by Slater and Ahuja [2] the need was felt to build a more general tutoring framework by combining features of different programs. This framework could be quickly used to build intelligent tutors for various problems without duplicating all the efforts. Or in other words, the need was felt to build some sort of authoring system shell with the help of which the subject expert would be able to create a variety of applications software.

With the perceptible shift in the availability of faster PCs at reasonable cost, it is imperative that more and more software should be made available on this platform. Obvious drawbacks of the PC-based application software are the memory restriction of DOS, the absence of graphic accelerator packages and the machine–user interaction interface. On the positive side, the rapid growth and wider acceptance of Windows and C along with associated libraries and other object-oriented languages in the 1980s was beneficial for the rapid development of authoring tools with varying

* Accepted 4 September 1994

capabilities. C++ satisfied all the requirements of conventional higher-level languages such as FORTRAN, allowed better memory management and provided excellent graphical and animation support.

REVIEW OF SELECTED PC-DOS AUTHORING SYSTEMS

An authoring program like STORYBOARD 2 for CAL is a flexible and useful text reconstruction program [3]. This is essentially designed for language learners. Text reconstruction involves knowledge of vocabulary and grammar syntax, cohesion and stylistic features. It works with MS-DOS and with pointer devices such as a mouse. However, this is predominantly a text-oriented program. Another tool that could be used for the purpose of authoring courseware is GRASP (GRaphics Animation System for Professionals), which is a programming language editor and interpreter [4]. GRASP creates, edits, plays program files and offers control over other system components. It also offers a Paintbrush-type utility for creating images, as well as an animation facility for creating special effects. It has a facility for linking external programs. Programs created by such a utility require large amounts of memory and would be more or less linear. QUEST is similar type of authoring system using the Quest Authoring Language with a Pascal-like syntax; this allows text, graphics and animation with conditional and unconditional branching. With the help of prompt-line text and a graphics generator one can build frames that can subsequently be linked. QUEST provides a rudimentary question-and-answer facility. The main emphasis of all these authoring systems is to create software consisting of text as well as graphics and to provide a suitable frame as well as an external program linking facility.

Multimedia authoring tools based on Windows

There are a number of authoring tools currently available that use Windows and its graphical user interface (GUI) platform. Most of these tools are quite versatile, providing facilities for audio, video, graphics as well as animation and text on a single platform. Broadly, the multimedia activities can be divided into tasks, events, actions and responses [5]. Each of these are the equivalent of subroutines or subprograms and hence can be developed independently. The screen layout is predominantly graphical in nature, allowing the author to produce materials easily with the help of a flowline approach of these activities, which are represented by icons. Such a method permits one to develop the application in a modular fashion. It provides user interaction with the help of click buttons, hot-spot areas or text inputs. These are general-purpose authoring tools that require a certain amount of training and mastery. One should select an appropriate tool to suit one's requirements. Each of them have a few strong points and a few weaker areas. For example, Authorware does not provide hypertext links, which are convenient for searching, whereas Microsoft's Multimedia Viewer provides better text search facilities but not a very elaborate GUI. Multimedia Tool Book uses pages, objects and scripts, rather than a flowchart approach, and these seem to slow down considerably when handling large amounts of data. These tools do provide theoretical facilities for linking with external programs. But they might not offer the best solution for handling extensive numerical work as is required in engineering disciplines. If one is not keen on using all the elements of multimedia, then one can use simpler approaches for producing the material.

Basic requirements of an authoring tool

Well-developed CAI/CAL application programs are interactive and user friendly. In order to achieve this, a substantial amount of effort—often nearly 40% of development time—is devoted to code development for an effective user interface. There is an inherent limitation on information which could be presented on a single screen with appropriate emphasis on text, pictures, equations and similar components. Organization and presentation control of materials becomes very important. To communicate the ideas pertaining to the subject matter effectively, these layouts must be prepared with the subject expert. To minimize programming efforts while concentrating on concepts, subject experts can use authoring tools. These tools should allow the necessary flexibility in creating the learning materials in a loose fashion in terms of frames or logical steps and provide a facility to link them logically.

Dym and Levitt list several issues that one must tackle to produce effective learning materials [6]. One pertains to the learning language, another to the operating system and still a third to the hardware platform. Most engineering applications programs such as finite element packages, or graphics/visualization packages are coded in procedural languages such as FORTRAN, Pascal or C. Knowledge-based systems in engineering (KBSE) modules, however, will be developed most easily in an artificial intelligence programming environment or with an expert system tool and shell, probably using Lisp or Prolog. One way to interface the two would be through a file transfer facility, which would essentially be a weak coupling that inhibits information transfer between the two systems. Another desirable way is to have engineering application programs and KBSE modules written in the same language. One middle ground that appears viable seems to be using of C++ for both.

The teaching of science or engineering subjects is different to that of language-based descriptive subjects. In science and engineering one explains basic concepts, writes equations, performs algebraic manipulations, simplifies them into alternative forms and utilizes these modified equations for some numerical evaluation. Thus the subject

matter dealt with in engineering education is often concerned with the ability to process various equations. Most of the subjects in the thermal and fluid sciences deal with fairly simple algebraic transformations which are used for optimization or sensitivity studies. Using the output of these equations, one can plot the numerical results on appropriate x–y graphs to study the trend analysis. In a classroom environment teachers may present a simple system with the help of schematic diagrams which are necessary for conceptual understanding. However, for complex systems they may use photographic slides, pictures, charts or a movie. The present work attempts to provide facilities to perform the tasks that teachers undertake in a regular classroom. In addition to the sequential form of information presentation, a hypertext/hyperpage facility is developed where context-sensitive or exploratory-type non-linear learning modes can be used by the learner.

MAIN COMPONENTS OF THE AUTHORING TOOL

ASEE (Authoring System for Engineering Education) has been developed with a limited number of components to provide the author with some tools for standard tasks. The materials produced will have a very close resemblance to classroom teaching. A limited objective of this work was to ensure a classroom-like appearance of the final product. This can be explained with the help of a few figures. All figures included in this paper were generated via the print screen utility. Thus these are the actual images seen on the screen (without the colors). A few of them have been altered slightly to suit black and white printing.

The information that the author wants to present is classified into 10 different categories. After entering a topic or an equation, an appropriate explanation is provided by the author for the same. Figure 1 shows a typical screen layout with all the possible options and a sample information entered by the author on one topic. Such information is created as ordinary text with the help of a built-in editor. The editor has standard edit functions such as insert, over(write), word or line delete, and automatic word-wrap. Appropriate data structures such as doubly linked lists have been used in the development of this editor for efficient and optimal use of memory. There is no restriction on the number of lines one can enter for the relevant text; however, it should be kept in mind that the text is equivalent to a teaching module, and so predominantly short and crisp information units should be made available to the learner.

Equations in science and engineering invariably include mathematical and Greek symbols. These are entered by the author in an ordinary text format as alpha, beta, pi, etc. Writing wquations in this way simplifies the author's work. In addition, text format equations are subsequently used for algebraic manipulations, number generation and plotting. However, equations in a textbook or on a classroom blackboard are not in alphanumeric text format. In addition to various mathematical symbols, one might be required to include subscripts or superscripts. For better visualization and appreciation, these equations should appear exactly in the same way as written in the classroom or in the textbook. Authoring systems using a Windows type of GUI are much more flexible in terms of type and size of fonts. However, these cannot be directly used for symbolic manipulations and numeric computations in an interactive fashion. This is illus-

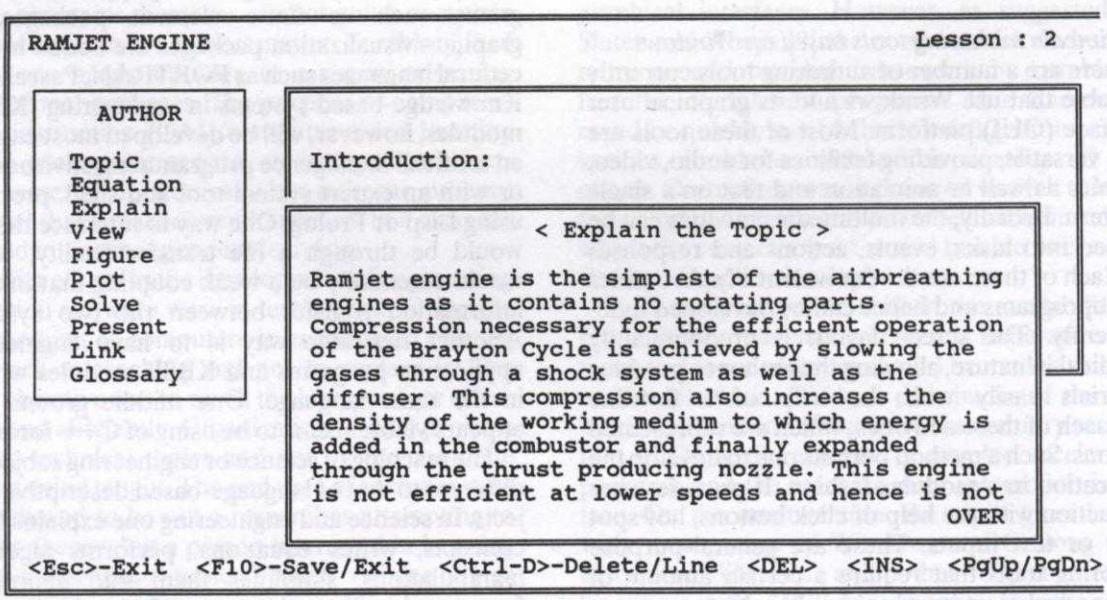

Fig. 1. Main components of authoring tool.

trated by the following example. For the sake of *mathematical* computation one writes a typical equation in a single line format as

```
pir = [1 + (gamac − 1)/2 * M0∧2] ∧ [(gamac − 1)/gamac]
```

The operator ∧ is used in this authoring tool for exponentiation; the other mathematical operators have their usual meaning. An equivalent expression in FORTRAN syntax would look like

```
pir = (1+(gamac−1)/2*M0**2)** ((gamac−1)/gamac)
```

Rapid comprehension of either of these relatively simple equations is not easy. For a teacher and student this equation is supposed to look like

$$\pi_r = [1 + \frac{(\gamma_c - 1)}{2} M_0^2]^{(\frac{\gamma_c}{\gamma_c - 1})}$$

Such a facility is included in this package by drawing commonly used Greek and mathematical symbols with the help of C language graphic primitives with appropriately scaled characters for superscript and subscript variables. Thus 'alfa' defines α, 'beta' defines β, 'gama' defines γ and similar commonly used symbols are predefined and character generated. The symbol names are at times short and cryptic to reduce the length of variable names. In addition, the author can define aspects pertaining to the subscripting and superscripting of variables with the help of a *glossary* facility. The glossary establishes relations between the variable name and entity which will be displayed on the screen and is shown in Fig. 2. The equations are displayed on the screen using a built-in autosizing facility for the main variable as well as the subscript and superscript characters. The author does not have to worry about the font type or font size. Admittedly this facility is not as extensive as those used in DTP packages or in professional authoring systems, but it is very simple to use and should be sufficient for the purpose of classroom-style teaching. Additionally it is more versatile from the point of view of multiple use both for display as well as mathematical manipulation or computation. These equations could be previewed by the author with the help of a *view* facility.

The author might like to include schematic or conceptual figures or pictures to explain a point during the course of a lecture. These are better than classroom blackboard pictures as one can use colors, etc., but are not of photographic quality—although they are sufficient to make a point. For this purpose a picture or figure drawing facility is provided with Paintbrush-type graphical tools. This facility is also constructed with the help of C language graphic primitives. It allows one to draw lines, circles, curves, closed figures, fill these with various patterns and with the standard 16 colors. It also allows the author to include or superpose text on the figure. One such schematic drawing is shown in Fig. 3.

In an actual textbook, a limited number of equations is included and the student learns to manipulate these equations for different purposes. Teachers generally demonstrate these operations in the classroom. Here an attempt has been made to provide a tool that would enable the user to manipulate the equations provided by the author. Such an approach is beneficial for an exploratory style of learning. For the programming, in the higher-level languages, one must write expressions where there is only one dependent variable in the simplest form on the left-hand side, and the rest of the constants and dependent variables are on the right-hand side. Depending on the requirements, the same equation could be written in an alternate form by simple operations of the opposite nature. Thus 'plus'

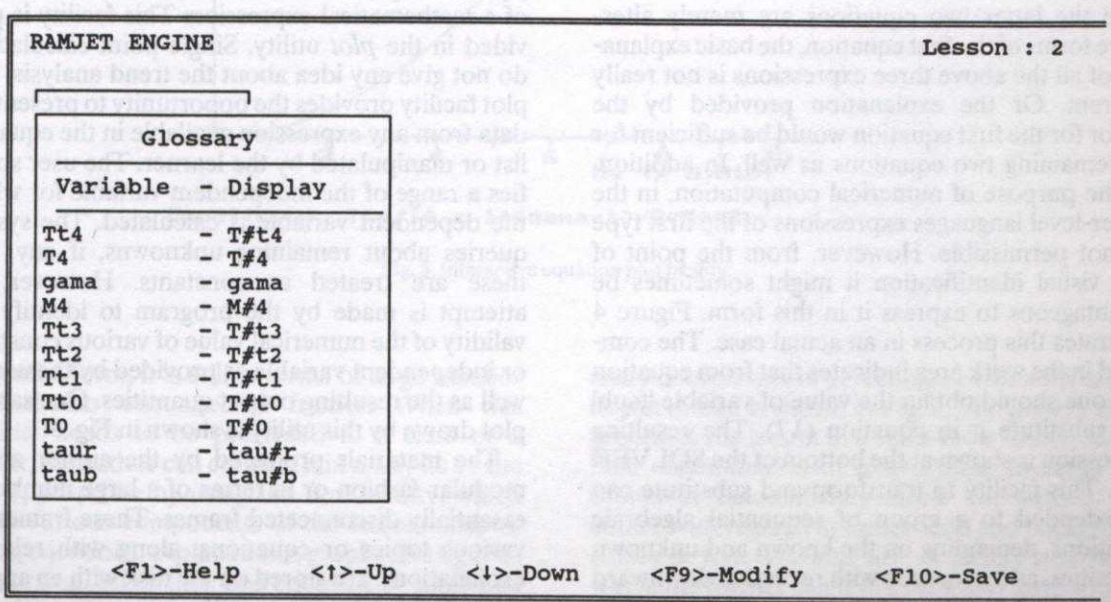

Fig. 2. Glossary facility for screen display.

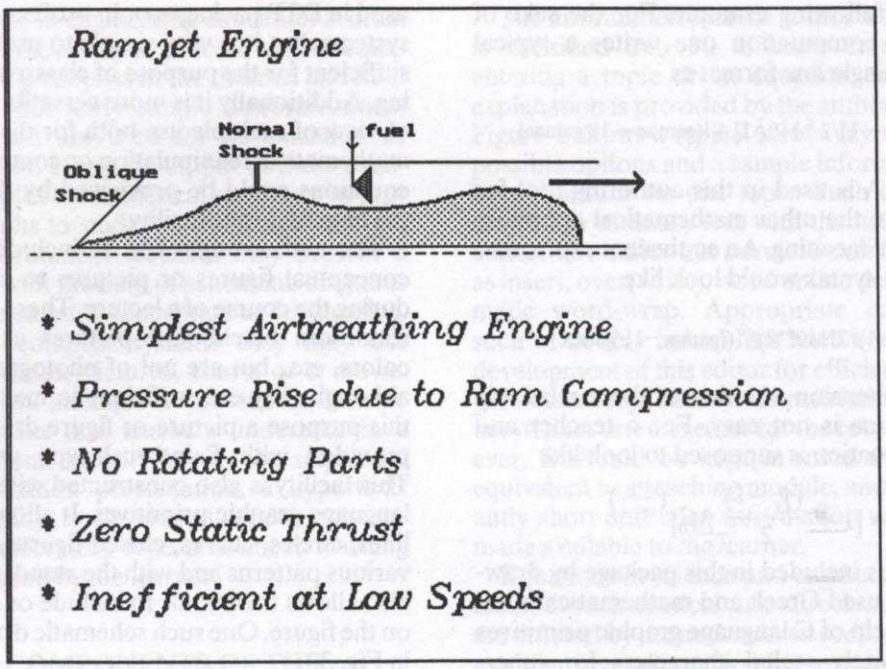

Fig. 3. Schematic diagram drawn by authoring system drawing tool.

becomes 'minus', and 'sine' becomes 'sine inverse' across the equal sign. This procedure can be explained by considering another very simple example

$$a^*x^2 + b^*y^2 = c$$

If one has to solve this expression for x, then one obtains in the algorithm the following expression

$$x = [(c - b^*y^2)/a]^{0.5}$$

Alternatively, to solve for b the same expression would look like

$$b = (c - a^*x^2)/y^2$$

Since the latter two equations are merely alternative forms of the first equation, the basic explanation of all the above three expressions is not really different. Or the explanation provided by the author for the first equation would be sufficient for the remaining two equations as well. In addition, for the purpose of numerical computation, in the higher-level languages expressions of the first type are not permissible. However, from the point of easy visual identification it might sometimes be advantageous to express it in this form. Figure 4 illustrates this process in an actual case. The command in the work area indicates that from equation (15) one should obtain the value of variable 'taub' and substitute it in equation (17). The resulting expression is shown at the bottom of the SOLVER box.. This facility to transform and substitute can be extended to a group of sequential algebraic equations, depending on the known and unknown quantities, as it proceeds with recursive downward parsing until no more equations can be solved. The resulting expression arising out of such a symbolic substitution may not necessarily be in the most presentable or readable form. However, a facility has been provided in the authoring system where the student can explore and learn by manipulating the equations, study them and draw his/her own conclusions. The numerical evaluation of such an equation is simply by merely substituting the numerical values for independent variables. This can also establish a link list between various frames other than the one proposed by the author and thus provide an option for a non-linear learning model.

Simple graphical representation of the final results for the user-defined variables in terms of x–y plots can give additional insight into the behavior of a mathematical expression. This facility is provided in the *plot* utility. Single-point calculations do not give any idea about the trend analysis. The plot facility provides the opportunity to present the data from any expression available in the equation list or manipulated by the learner. The user specifies a range of the independent variable for which the dependent variable is calculated. The system queries about remaining unknowns, if any, and these are treated as constants. However, no attempt is made by the program to identify the validity of the numerical value of various constants or independent variable as provided by the user as well as the resulting output quantities. One sample plot drawn by this utility is shown in Fig. 5.

The materials produced by the author are in modular fashion or in terms of a large number of essentially disconnected frames. These frames for various topics or equations, along with relevant explanations, are stored on the disk with an appropriate link list tag. Though the materials written by the author would by and large be in a logical and

```
RAMJET ENGINE                                          Lesson : 2

                                  -= SOLVER =-
(15) f = (cp*T0*taur)/h*(taub-1)
(16) Isp = Fs / (g*f)
(17) taulamda = taub * taur
(18) T4/T0 = taub



(17) taulamda = ((f/((cp*T0*taur)/h))+1)*taur)

                                -= WORK AREA =-
command : disp
command : 15>taub=>17
command : quit
```

Fig. 4. Algebraic symbolic solver.

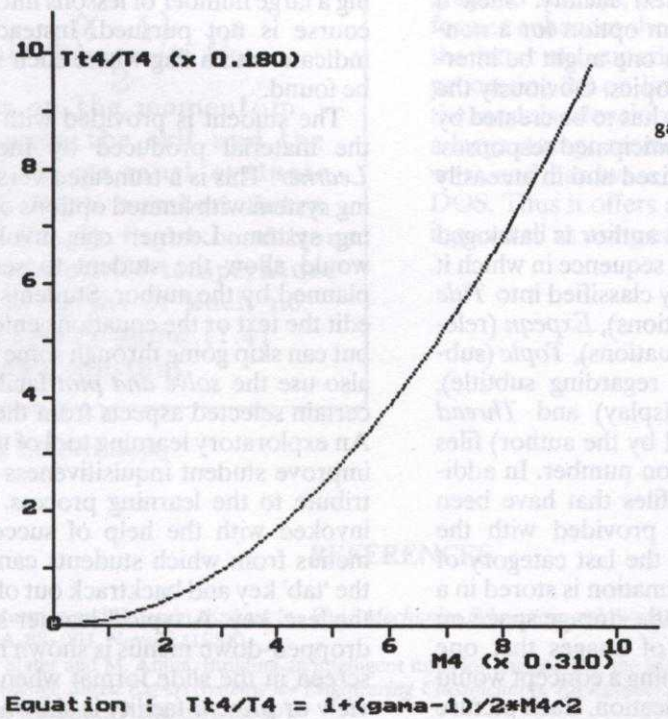

Fig. 5. Interactive equation plot facility.

sequential form, it is still in terms of large number of dissimilar unconnected frames. When this material needs to be presented to a class or a learner, the author can connect it in a way he or she feels appropriate by using the *link* facility, which allows various components of the lesson, such as topics, equations, explanations, graphs and figures, to be linked. More than one equation or topic and corresponding explanation combination could be accommodated in a single screen by specifying x and y coordinates in percentages. One such sample lesson thread is shown in Fig. 6. The *present* utility generates the lesson in a slide-show format for the final dissemination of information. Material presented is automatically sized but without any choice with regards to foreground and background colors or font type and size. A facility is provided where the learner can move back and forth with the help of the page up/page down facility. Alternatively these frames could be linked with the help

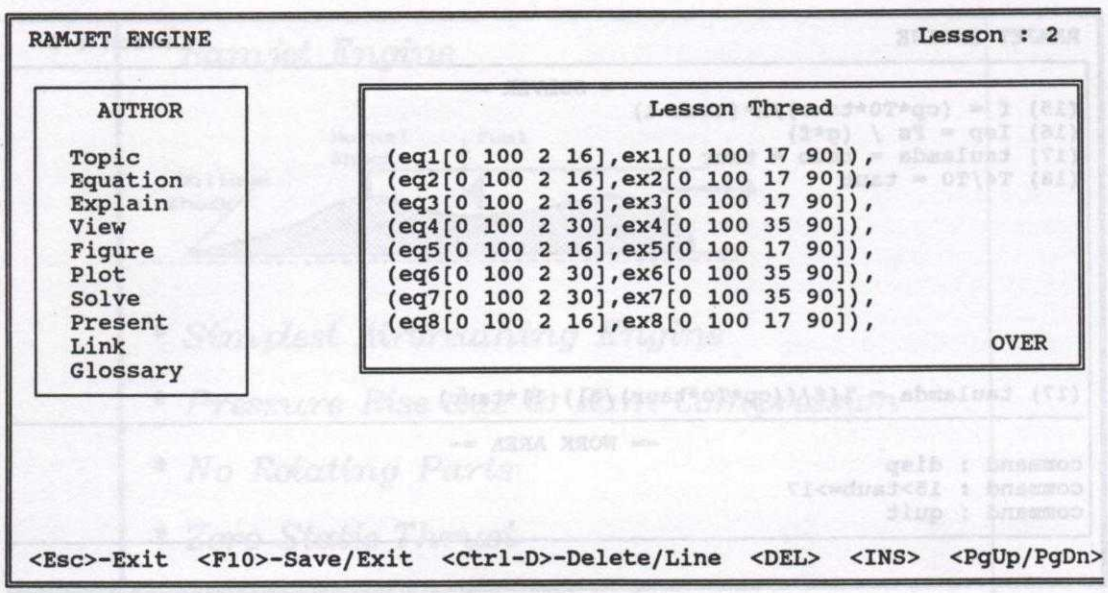

Fig. 6. Lesson thread facility for author.

of a context-sensitive *hypertext* facility. Such a facility would again provide an option for a non-linear mode of learning where one might be interested in a limited number of topics. Obviously the hypertext or hyperpage facility has to be created by the author depending on the anticipated responses. Materials presented are autosized and in an easily readable form.

Information entered by the author is cataloged by the authoring system in the sequence in which it was entered. It is automatically classified into *Title* (main lesson title), *Eqn* (equations), *Expeqn* (relevant information regarding equations), *Topic* (subtitle) , *Exptop* (information regarding subtitle), *Table* (glossary table for display) and *Thread* (lesson sequence as envisaged by the author) files with the extension of the lesson number. In addition, one can include figure files that have been drawn by the drawing tool provided with the authoring system. Except for the last category of figures, all the remaining information is stored in a text form that requires very little storage space on the hard disk. The number of images that one would like to utilize for explaining a concept would obviously depend on the application. Each picture frame would need about 110 kbytes of storage space in uncompressed mode. The picture library generated for related topics could be shared by more than a single module. When the rest of the information is viewed as a single frame or seen as a slide-show through the present facility, it is exploded with the built-in program. A *solver* or *plot* type utility is indeed an interpreter-type program. The main advantage of the program-like structure is that the execution speed is rather fast when compared to the sequential search in a larger database. The author can produce one module covering a single lecture with the help of this authoring tool. For the time being, the idea of linking a large number of lessons into a single complete course is not pursued. Instead the author can indicate with a flag where such information could be found.

The student is provided with an interface with the material produced by the author through *Learner*. This is a truncated version of the authoring system with limited options of the main authoring system. Learner can invoke *present* which would allow the student to see the material as planned by the author. Students obviously cannot edit the text or the equations entered by the author but can skip going through some of them. They can also use the *solve and plot* facility to learn about certain selected aspects from the complete lesson. An exploratory learning tool of this type is likely to improve student inquisitiveness and thereby contribute to the learning process. Such a facility is invoked with the help of successive drop-down menus from which students can select with using the 'tab' key and backtrack out of it with the help of the 'esc' key. A typical learner screen layout with dropped-down menus is shown in Fig. 7. The same screen in the slide format when seen through the *view* or *present* facility is shown in Fig. 8. A complete presentation utility would include the number of screens being presented in an order prescribed by the author. The learner has the option to view only those screens in which he/she is interested

CONCLUSIONS

An attempt has been made to develop an authoring system, ASEE, that would be well suited to developing materials for teaching engineering subjects. The main objective was to put together selected and limited parts of complete multimedia authoring systems, which rely heavily on text and

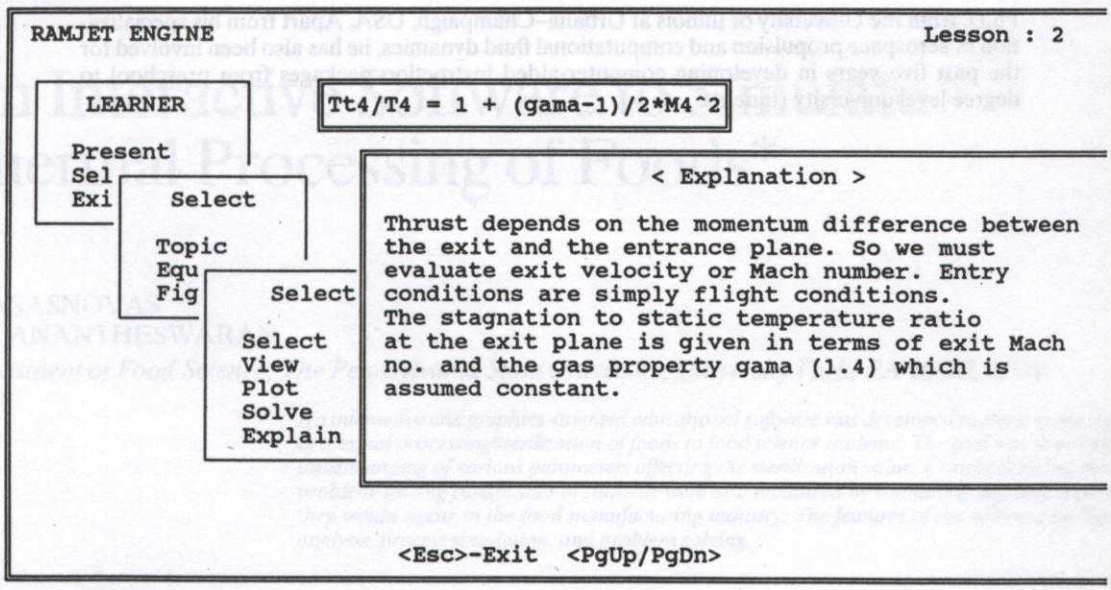

Fig. 7. Drop-down menu for learner.

$$\frac{T_{t4}}{T_4} = 1 + \frac{(\gamma - 1)}{2} * M_4^2$$

Thrust depends on the momentum diffference between the exit and the entrance plane. So we must evaluate exit velocity or Mach number. Entry conditions are simply flight conditions. The stagnation to static temperature ratio is given in terms of Mach no. and the gas property gama (1.4) which is assumed constant.

Fig. 8. Present facility.

graphics screens, with application programs. The former enhances the presentation quality, whereas the latter are superior in terms of fast numerical processing for on-line output information. Due to the modular development, ASEE does not require a large storage memory, or any other special software and hardware beyond an ordinary PC with DOS. Thus it offers a low-cost option for producing as well as using educational software.

REFERENCES

1. M. Murman and R. Lavin, *Enhancing Fluid Mechanics Education with Workstation-based Software*. AIAA-88–001, Nevada (1988).
2. J. H. Slater and M. Ahuja, Building an intelligent tutor for engineering: the Maccavity experience. In *Knowledge Based Expert Systems for Engineering Classification, Education and Control*, Computational Mechanics Publication (1989).
3. C. Jones, *STORYBOARD 2: An authoring Program for Computer Assisted Learning*. Wida Software, London/Eurocentres, Zurich (1992).
4. J. Bridges, *GRaphics Animation System for Professionals*, Version 4.0. Paul Mace Software (1991).
5. Multimedia development, *PC Magazine*, pp. 151–199 (August 1994).
6. C. L. Dym and R. E. Levitt, *Knowledge Based Systems in Engineering*. McGraw-Hill, New York (1991).

P. P. Ganapathi Subramaniam completed his bachelor's degree at Coimbatore in India with a mathematics major. After completing a masters in computer applications he worked in a software company for couple of years. Currently he is registered for an advanced degree program in the Department of Computer Science and Engineering at IIT Madras, India.

S. S. Gokhale is an Associate Professor in the Department of Aerospace Engineering at IIT Madras, India. He received his B.Tech. and M.Tech. from IIT Bombay, India and MS and

Ph.D. from the University of Illinois at Urbana–Champaign, USA. Apart from his specialization in aerospace propulsion and computational fluid dynamics, he has also been involved for the past five years in developing computer-aided instruction packages from preschool to degree level university students.