Courseware Development: A Comparison of Three Programming/Authoring Environments

Thomas E. Ludwig, Hope College

Courseware developers can choose from a variety of programming environments on the Macintosh, ranging from standard programming languages to special-purpose tools for creating instructional modules. This paper compares three representative development environments: ZBasic, a powerful BASIC compiler thal offers total control of the Mac interface; HyperCard, the standard for courseware development; and Authorware Professional, the most powerful (and most expensive) of the instructional development systems. A sample courseware module illustrating the structure and function of the human auditory system (taken from the PsychSim courseware package), developed in parallel in these three environments, will serve as the basis for the comparisons. The strengths and weaknesses of each environment will be discussed from the developer's point of view and from the student user's point of view.

When I began developing instructional software in the early 1980s, courseware authors had little choice but to use a general purpose programming language such as Pascal or BASIC. At the time, these languages were not at all conducive to courseware development. The available dialects of BASIC were unstructured and lacked power, and the early versions of Pascal didn't even support graphics! Fortunately, things have changed. The BASICs began to support structured programming, C entered the scene, and most of the general-purpose programming languages have moved toward object-oriented programming.

But the most exciting change for instructional developers has been the appearance of completely new options for courseware development, especially on the Macintosh. Some of these new development environments are specifically designed for producing courseware, while others are general-purpose tools that can be easily applied to courseware development. As I see it, the options for the courseware author fall into three categories:

• General Programming Languages such as Pascal, C, or BASIC, along with their object-oriented varieties.

• Hypertext Development Environments such as HyperCard, SuperCard, Guide, and PLUS.

• Specialized Authoring and Multimedia Development Environments such as Director, Filmmaker, MediaTracks, and Authorware.

This paper will examine a representative development environment from each of these three categories. From the general programming languages, I've selected ZBasic by Zedcor, Inc; from the hypertext environments I've selected HyperCard by Claris, Inc.; and from the specialized authoring tools I've selected Authorware Professional by Authorware, Inc. I'll analyze the advantages and disadvantages of these specific environments for courseware development, and I'll try to make some general statements about the usefulness of each category of development tools.

These comparisons will be based on one particular module from my PsychSim courseware package for introductory psychology courses. The PsychSim package (which won the 1990 EDUCOM/NCRIPTAL Higher Education Software Award for "Best Psychology Software") contains 16 modules covering a range of topics in psychology. The module I've selected is titled "The Auditory System," and focuses on the structure of the ear and the process of hearing.

In order to make a fair comparison of the three programming environments, I developed an instructional module in each of them in parallel, using the same content in all three. The particular software versions used were ZBasic 5.0,

HyperCard 2.0, and Authorware Professional 1.6.

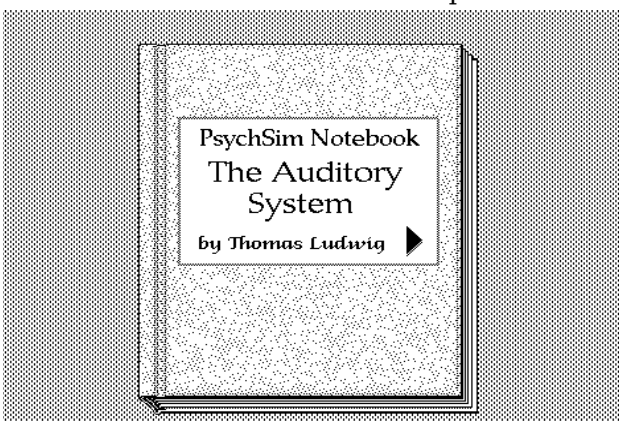

Figure 1. *Title page for auditory module.*

I have grouped my observations about these environments into eight areas: Cost, The Learning Curve, Availability of Help, Program Logic, Prototyping, Testing and Debugging, Production and Distribution, and Use by Students.

Cost

ZBasic and HyperCard are both fairly inexpensive. ZBasic is available by mail order for approximately $100. For comparison, Microsoft QuickBasic is about $65, and THINK C and THINK Pascal by Symantec each cost about $165.

A minimal HyperCard environment is shipped with each new Macintosh, while the HyperCard 2.0 upgrade is (or was) available directly from Claris for about $50, and the HyperCard Developer's Kit can be purchased for about $130 by mail order. For comparison, Silicon Beach's SuperCard 1.5 costs about $200, and Spinnaker's PLUS 2.0 is about $290.

The high end of the spectrum is anchored by Authorware Professional 1.6, listing at $8000 but available to educators for about $1000. The other specialized tools are also fairly expensive, with mail order prices averaging about $450 for Paracomp's FilmMaker 2.0 and MacroMind Director 2.0, and about $200 for Farallon's Media Tracks.

The Learning Curve

Like all the standard programming languages, ZBasic is a fairly difficult environment to master for those without prior programming experience. The syntax is complicated and precise, and the programmer really needs to know something about the internal workings of the Macintosh interface. The tremendous power of the language, especially its ability to use any of the Macintosh toolbox calls and even to alter the contents of the CPU's registers, means that inexperienced programmers can crash the system in hundreds of ways. This power and flexibility also produces a slow learning curve. As an example, even though I had about 8 years of previous programming experience with a number of languages including several dialects of BASIC, it still took me about a year to become proficient in ZBasic.

In contrast, HyperCard has a fast learning curve. I have seen individuals with zero programming experience produce usable stacks after only two or three days of training. Of course, HyperCard's real power lies in the HyperTalk scripting language, and that takes more time to master. But the modularity of the script handlers and the ready availability of sample scripts from other developers makes HyperTalk one of the easiest programming languages to learn. In the space of several weeks I moved from a HyperCard novice to a "power user."

Authorware Professional has a learning curve somewhere between ZBasic and HyperCard. Its visual flow-chart organization (see below) and its HyperCard-like drawing tools make it easy for inexperienced developers to produce simple interactive modules. However, its considerable power, its rather complex scripting language, and its ability to create sophisticated animations stretch out the learning curve for serious developers who want to make use of all its features. Authorware offers special training seminars for its customers, but I estimate a relatively knowledgeable developer could master Authorware on his or her own over a period of several months.

Availability of Help

All three of these environments have substantial vendor support, in the form of telephone help lines available to registered users. Zedcor and Claris provide free support, while Authorware charges an annual fee of $150. Authorware also provides a quarterly newsletter with tips for developers. However, the most significant help that Authorware provides is a library of modular routines and sample courseware ideas. This

library is included in the development package and is well-indexed for easy use. Zedcor also provides a set of example files, and Claris includes several stacks of programming ideas, but these samples aren't as extensive or well-documented as the ones Authorware provides. On the

other hand, both ZBasic and HyperCard have on-line Help files that can be accessed during program development, while Authorware does not (I can guarantee that every Authorware reference manual will be well-worn in a few months!). Finally, HyperCard's popularity and wide use put it in a special class: There are dozens of books and third-party add-ins to help the HyperCard developer, as well as hundreds of public domain or shareware stacks containing tips or development tools.

Program Logic

In many respects, Authorware is a programmer's dream come true. Its development environment consists of a multi-window flow chart, with program segments represented by small icons.

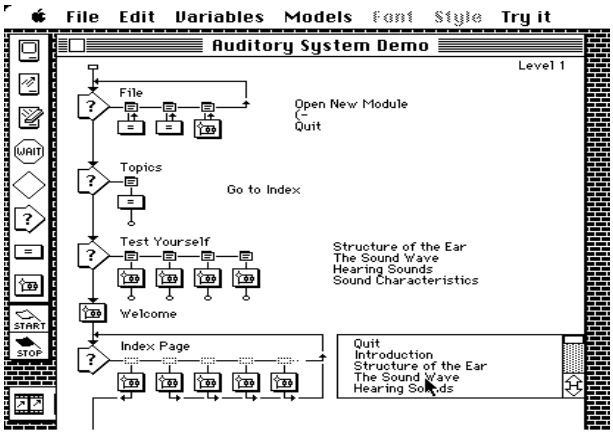

Figure 2. *Sample Authorware developer's screen.*

These icons, shown on the vertical bar at the left in Figures 2 and 3, represent various program elements. Since most Mac programmers are not familiar with Authorware, let me digress a bit to explain Authorware's program logic. The small image of a Macintosh at the top of the vertical bar is a Display icon which, like a HyperCard card, can contain a single text field or graphic object, or many such items. The next icon is an Animate icon, which contains instructions for the animation of one of the objects previously placed on the screen by a Display icon. The object can be moved along a fixed or variable path using one of five different styles of animation. The third icon is the Erase icon, which removes the contents of the designated Display icon from the screen. This allows selective erasure of some objects without disturbing other portions of the screen. Next comes the Wait icon shaped like a stop sign. It produces a pause which can be programmed to last a certain number of seconds, or until the user presses a key or clicks the mouse. The diamond shape is the Decision icon, which branches according to the value of a variable, or can be set to step sequentially through each branch of its decision tree, as in the case of Figure 3.

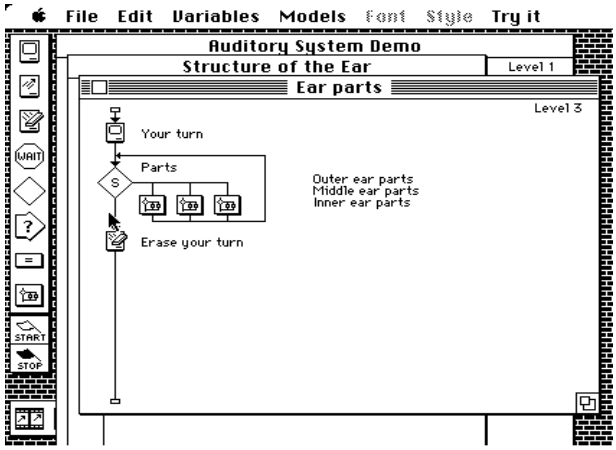

Figure 3. *Lower-level (nested) Authorware routine showing sequential selection of program segments.*

The question mark symbol is the Interaction icon, which poses a question or option, obtains a response from the user (through one of eight response modes), and branches accordingly, keeping track of the user's response and scoring it as correct or incorrect, if desired. The equals sign is the Calculate icon, which contains program script to perform calculations, set the values of variables, or perform other functions under program control. The next symbol is the Map icon, which merely contains a grouping of the other icons. If you click on a Map icon, a new window opens to reveal the icons contained within it. This is the method used to nest subroutines within the main program flow. Below that, you see the Start and Stop flags used for testing only a portion of the program. At the bottom are the multimedia icons representing PICS-type "movies" created by flipping through a series of still images, and sound files.

Program development in the Authorware environment consists mostly of dragging icons onto the flow chart, then opening the icon and adding the appropriate text, graphics, sound, animation, or branching logic for that portion of the program.

In the HyperCard environment, program development consists mostly of creating new blank cards, filling those cards with the desired information, and then creating the links to navigate from one card to other cards or other stacks. So in a typical development session, the Mac screen's appearance would be very similar to the way it would look while a student was using the module for instruction.

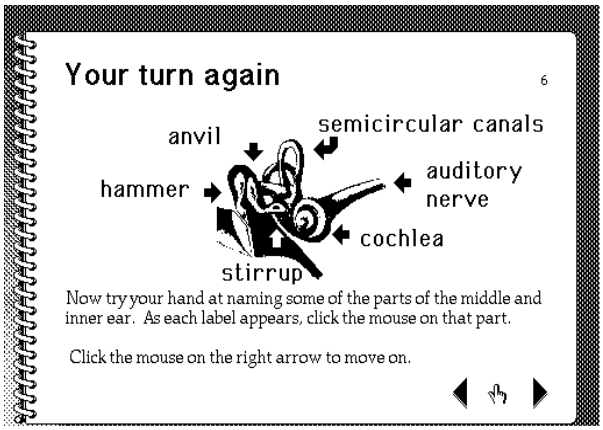

Figure 4. *Sample HyperCard screen.*

Of course, the developer would likely be switching back and forth between the Browsing mode (which the student would use) and the Button and Field and Paint modes, in which the various text fields, buttons, and graphic objects could be moved and manipulated. Also, the developer would probably make frequent reference to the scripts attached to the objects on the card (or even to the card itself) in order to modify the program logic that controls the student's navigation from one card to the next, as well as the actions of objects on that card.

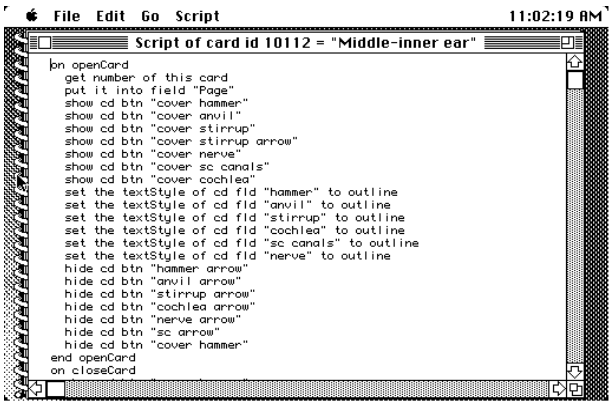

Figure 5. *HyperCard script for card shown in Fig. 4.*

The program logic used by ZBasic or any of the standard languages is quite different from both HyperCard and Authorware. Since ZBasic is only partially object-oriented, the program flow generally steps through the program statements in a linear fashion, with occasional function calls and branches to subroutines. The typical development session involves staring at a text editor screen full of semi-English statements, tweaking the values of variables to improve the placement of objects on the screen. But in order to actually see those objects, the program needs to be compiled and executed... and then edited again.

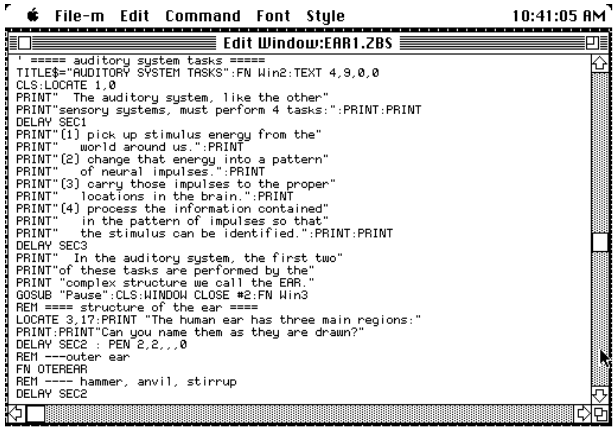

Figure 6. *Sample ZBasic program code.*

The fact that ZBasic offers total control of the Mac interface is both a blessing and a curse, because it means that the developer must specify the precise coordinates of every object on the screen, and must redraw them every time the screen needs to be updated and dispose of them when they are no longer needed. These details are handled transparently in HyperCard and Authorware, leaving the developer more free to concentrate on the instructional content

Prototyping

One of the most striking differences between ZBasic (or any standard language) and the new development environments lies in the ease with which new ideas can be translated into working prototypes. The ZBasic developer needs to have a considerable portion of the program code in place and debugged before anything will appear on the screen. Usually weeks of effort are expended before a working model is ready to be shown to a publisher or to student users for feedback. In contrast, a HyperCard developer can start from an empty stack and create a working prototype of most types of instructional materials in the space of a few hours, and this could shrink to less than an hour if the developer is allowed to use an existing stack as a model. The same is true for the Authorware developer: Ideas can be prototyped and modified again and again in the length of time that it would take a ZBasic programmer to bring up the first screen.

Testing and Debugging

When it comes to serious debugging tools, ZBasic and other standard languages usually have the edge, since they generally have built-in debuggers for tracing program flow and single-stepping through critical routines. HyperCard and Authorware have more limited debugging facilities, but this is not necessarily a problem. The simplicity of HyperCard and the modular structure of the HyperTalk event handlers mean that programming errors should arise less frequently, especially if the developer reuses scripts that have already been tested. The clear flowchart structure of an Authorware program makes it less likely that errors in program logic and branching will occur. In addition, Authorware provides easy-to-use “Start” and “Stop” flags which allow the program to be tested in small logical segments.

Production and Distribution

Once the testing is complete, the developer needs to decide how to package the program and distribute it for use. In the HyperCard environment, final production involves simply naming the stack and copying it to a disk. The developer has to assume that the end user has the proper version of HyperCard, since the stack cannot be compiled into a stand-alone application unless you purchase a third-party utility such as Compile-It by Heizer Software. The developer has little chance of retaining control of his or her program script, even if the Protect Stack option is used, since a serious hacker can easily break into the code.

In the Authorware environment, the developer has two options for packaging the program. If the program is a single unit, the developer will probably choose to compile the Authorware program and bundle it with the run-time module to create a stand-alone application. Unusual fonts can also be bundled into the application, if necessary. If the program has several modules, the developer will probably choose to include the run-time module separately on the distribution disk rather than bundle it with each instructional module, thus saving disk space.

With ZBasic, the developer can compile the program as a full-fledged Mac application, complete with its own application icon. Alternatively, the developer may choose to compile related modules as chain files which can be accessed through a menu-type application that contains the run-time routines.

The developer must also consider the size of the packaged program. ZBasic produces compact applications; the compiled Auditory System module takes up less than 60K. In contrast, the equivalent information packaged as a HyperCard stack takes up about 130K, and the Auditory System module packaged as an Authorware application (bundled with the hefty run-time routines) is over 300K.

Use By Students

All three of these development environments can produce instructional modules that are easy to use and robust. However, since ZBasic compiles programs into tighter, more efficient code, ZBasic instructional modules tend to run faster and are thus less frustrating to use on the older Macs. This is particularly true if the Mac doesn’t have a hard disk. On the other hand, many students are already familiar with the HyperCard environment and thus may feel more comfortable with an instructional module developed in HyperCard. Authorware has a slight advantage over the other environments when it comes to interactivity, in that it offers more different types of possible responses by students. In addition to pushbuttons, text entry into fields, single keypresses, mouse

clicks on hot words or graphics, Authorware allows students to make a response by dragging an object from one area to another, as in the case of assembling a molecule from its atoms.

Conclusions

All three of these development environments can produce high quality instructional modules, but that doesn’t mean that the environments are functionally identical.

Because of its ease of use, I'd recommend HyperCard to would-be academic developers without much programming experience. I'd also suggest HyperCard as the environment of choice whenever the information to be presented has a consistent structure. For example, a module on wildlife habitats might contain 80 sets of three elements: A picture of an animal, a map showing its geographical distribution, and a text field describing the animal's habitat. HyperCard would be ideal in this situation. Finally, because HyperCard stacks are not compiled, I'd recommend HyperCard for any instructional program that expected the end user to add to or modify the information in the module.

On the other hand, HyperCard is a poor choice for a project that requires extensive animation. HyperCard's animation options are limited and SLOW (although more powerful add-ins for animation, such as Add Motion by Motion Works, are now available). For simple, fast animation, ZBasic is a better choice, while for sophisticated path-based or data-driven animation, or animated "movies," Authorware is clearly superior. I'd also recommend Authorware for modules that include study guides or scored responses, because Authorware has over a hundred system variables and built-in functions that keep track of the user's performance, making it easy to give custom feedback or branch to remedial material.

If the instructional project includes the use of videodisc or CD-ROM materials, both Authorware and HyperCard have significant advantages over ZBasic, both in power and ease of use. Although ZBasic can control these devices, the programming is tedious and error-prone. Authorware is designed to support virtually all the standard videodisc players in a fully-integrated way, while HyperCard developers must obtain special drivers from Apple or other vendors.

Although I cut my teeth on standard programming languages, I now see their role in the future of instructional computing as mainly limited to certain situations in which small, fast modules are needed. In fact, since both Authorware and HyperCard can call external compiled routines, even developers who are proficient in a general programming language may find themselves using an authoring environment to control the general flow of the program, and merely using the programming language to produce small routines for speed-critical tasks.

As you can see, the tools for academic developers have improved a great deal in the past ten years, and they will undoubtably get more and more powerful and easy to use. I am genuinely excited about the future of instructional computing. I believe that we are about to see a flood of high quality, genuinely effective educational packages. The new hardware platforms and the new software development environments have finally brought the long-promised educational revolution within reach.