

Design Issues in a Stand Alone Multimedia
Computer-based Mathematics Curriculum

Raymond Ravaglia
Education Program for Gifted Youth, Stanford University

June 1, 1995

For the last five years the Education Program for Gifted Youth (EPGY) at Stanford University has used its stand alone multi-media computer-based courses to teach advanced placement calculus and physics to over 100 advanced middle school and early high school students. EPGY currently offers students several years worth of instruction in mathematics and physics beyond the advanced placement level. In this essay I will discuss several design principles which have come as hard won truths over the past six years of developing courseware at EPGY.

Before discussing these principals it should be noted that the EPGY software, unlike traditional applications of computers in education, is intended to be the primary means of instruction, and not merely to supplement a regular class. In fact, it is precisely those settings where a regular class cannot be offered, either because of an insufficient number of students or the absence of a qualified instructor, that the software is most ideally suited. It is towards computer-based courses in this sense that this discussion is aimed, and not towards educational projects which attempt to use computers to supplement some aspect of an a human taught course.

Informality plays an essential role

It has been often noted, though never adequately researched, that oral lectures are an important part of learning the mathematical and physical sciences. There is a natural difference in style between the way these subjects are written and the way they are talked about. The written style tends to be formal, detailed and rigorous, while the informal spoken style is more intuitive and also more flexible.

Multimedia presentation affords a unique way of embodying both styles of instruction. Traditional text dominated computer-based instruction has tended to follow the formal style of text books. This is understandable since anything which appears as writing is subject to standard conventions. Attempts to preserve an informal style in written instruction too frequent perish at the hands of editors.

By capturing sound and images outside of this limited context, multimedia provides a medium in which it is acceptable to preserve the informal style of the lecture, even though it is being published.

The question that arises is how to best capture what is presented in a lecture. The solution which comes readily to mind is video. But this must be rejected since, quite simply, video is vanity. It is not the image of the content provider, but the content itself, that is essential to a lecture. In abstraction all that matters is the lecturer's voice and the writing upon the board. This can be captured by using synchronized digitized sounds and graphics, either as a bitstream of what appears on the board, or as the equivalent of slides on an overhead projector. In this way

one preserves the essential features of the lecture: one has the instructor's voice and the writing on board; the only thing missing is the instructor.

In comparison, video suffers from serious drawbacks. It requires significantly more digital storage space than audio does. Moreover, the production costs of video are significantly greater than those of sound. Finally, it is a simple fact that the graphic images of handwriting are much sharper than what can be obtained by ordinary video shots of the lecturer and the chalkboard.

By focusing entirely on what corresponds traditionally to the chalkboard we are able to produce a fine, detailed image from the graphic tablet. Moreover, by avoiding video and working with synchronized sound and graphics, we are able to directly import into our lectures, images created in other software packages. Thus if we want to show students the graph of a surface, we can draw the graph using a professional package like Mathematica and import it directly into the lecture with no loss of quality. Lastly, this style also lends itself to natural synchronous extensions using "shared whiteboard" conference software.

Individualization v. the notebook model

The single greatest advantage that computers have as an instructional tool is their ability to individualize instruction to meet the needs of a particular student. This principle, recognized since the earliest drill and practice mathematics programs, remains essential, for while a teacher lecturing must address an entire class at once, a computer, by assessing each student's understanding, and tracking that student's performance over time, can adjust the level of instruction to match that particular student's needs.

Ideally both the amount of material presented to a given student and the level at which it is presented should vary according to the rate at which the student masters the material. Students who can move quickly through a given subject should be allowed to, while those who need additional work or a more concrete style of presentation should have those resources available to them. To accomplish this within a computer-based course requires assessment of a student's understanding at each point in the course, together with a model of that student's performance through the course up to that point. The judicious use of symbolic computation makes this type of assessment possible.

This type of assessment and the general pedagogical potential afforded by this ability to individualize instruction have been too often ignored by curriculum designers carried away by their excitement over the features embodied in sophisticated symbolic computation programs like Mathematica. The reason for this oversight lies in the inherent limitations of the "Mathematica Notebook" approach to

course development.

In the notebook model students are presented with what essentially are “dynamically unfolding objects.” Students can explore these objects and interact with them in real time. One can click on the equation of a function and produce its graph, rotate it in three space, and automatically evaluate its integral. Or one can investigate a functions limits by creating a table of values with arbitrarily great decimal precision. While such experimentation has its place in a computer-based mathematics course, it is not sufficient in itself to constitute a course.

The biggest problem with those who decide to develop courses within the confines of the “Mathematica Notebook” model is that it is fundamentally a static, non-adaptive model since the program does not assess student performance with the idea of modifying the style of presentation in any way. Whether or not a student has understood the concept being presented on page 4, will have no effect on the subsequent presentation of the course. Consequently, the interaction the students have with these notebooks occurs independently of what their degree of mathematical sophistication is. This is not only a poor utilization of a powerful resource, it is exactly the wrong way to approach instruction.

Assessment: correct uses of symbolic computation

To be responsive to a student’s level of understanding, a computer-based course must be able to assess the student’s comprehension of the material on at least two levels. The first is to determine whether or not the student is able to produce the correct answer to the sorts of questions that the student will see on examinations. The second is to determine whether or not the student who produces correct answers actually understands why the answers are correct.

Assessing whether or not students can produce correct answers is relatively straightforward and can be done by asking students the sorts of questions that instructors traditionally ask after lectures or on examinations. However, in designing such free answer questions several issues must be taken into consideration.

The most basic is the ease of input. If a student has to type a complex mathematical expression in an input language, the odds that an incorrect response is caused by an error in typing will exceed the odds that it is caused by an error in understanding. Care must be taken to provide the student with a convenient means of input such as typesetting programs do, together with the ability to see their input formatted, so that they can verify that what the computer has understood is in fact what they wished to express.

Equally important is naturalness of input. Students should not have to constrain their answers to fit a particular form, outside of those constraints which

an instructor would reasonably place upon them in class. While it is reasonable to require fractions to be expressed in lowest terms, it is not reasonable to prefer prime notation to Leibniz notation.

Inflexible processing of student answers is a standard complaint with computer taught courses. Because answer comparison is most often some variation of string matching, students are required to follow a standard form for input. Unfortunately this eliminates the wide variety of natural variations in the ways people express even relatively simple mathematical expressions.

The correct approach is to process the answers symbolically, taking into consideration their mathematical meaning, and considering possible correct answers in terms of equivalence classes. This minimizes the need to require that students conform to an arbitrary input standard, allowing the computer to understand natural variations of correct answers, and thereby accommodating different approaches to the solution of a problem which can result in equivalent answers with different forms to be evaluated as correct responses.

A simple example from algebra shows the natural variety that a student's answer can take. Suppose a student is asked to solve the equation $x^2 + x + 1 = 0$ in the complex plane. One may want to accept as correct all of the following variants: $\frac{-1+i\sqrt{3}}{2}$ and $\frac{-1-i\sqrt{3}}{2}$; $-\frac{1-i\sqrt{3}}{2}$ and $-\frac{1+i\sqrt{3}}{2}$; $\frac{-1}{2} + \frac{i\sqrt{3}}{2}$ and $\frac{-1}{2} - \frac{i\sqrt{3}}{2}$; $\frac{-1}{2} + i\frac{\sqrt{3}}{2}$ and $\frac{-1}{2} - i\frac{\sqrt{3}}{2}$, not to mention several others with essentially the same form. (If variations in spacing are considered, the number of alternatives increases even more dramatically.) To code each of these pairs of answers for the purposes of simple string comparison would be a chore and would fail to exploit the semantic content of the mathematical expressions. What counts as a correct answer here is any pair equivalent under certain transformations to $\frac{-1+i\sqrt{3}}{2}$ and $\frac{-1-i\sqrt{3}}{2}$. Whether or not the student's answer is correct can be determined by passing the student's input and the author coded answer plus specification of equivalence class to a symbolic computation program for evaluation and comparison. Exploiting the fact that the answers are mathematical expressions increases the flexibility for student input and simplifies author coding.

The more difficult assessment task is to determine if a student has actually understood the question at hand. While getting the correct answer is the goal of solving a problem, getting it in the correct way is equally important. Hence the mathematics teacher's dictum "Show your work."

The EPGY software has addressed this issue by creating a derivation system, i.e. an environment in which students can formally manipulate mathematical expressions by applying inference rules. A derivation system differs from a symbolic computation environment by having the logical structure necessary to represent

mathematical inference and logical dependency. This enables the derivation system to detect when students make fallacious inferences while working a problem. In the environment, the student supplies the rule and the derivation system takes care of performing the appropriate calculation. The results of the calculation are preserved for the student to further manipulate. A derivation of a problem is the set of steps from the statement of the problem to the solution.

The derivation system plays a crucial role in moving beyond the traditional limitations in evaluating student understanding. Traditional computer instruction evaluates a student's performance solely in terms of whether or not he or she produces correct answers to questions asked. It does not examine the process that the student went through to produce the answer. As such, it provides an inadequate assessment of this second level of understanding. Derivation systems, however, by requiring students to explicitly justify their inferences, make it possible to thoroughly examine a student's understanding of the rules and theorems underlying a given body of mathematics. Furthermore, derivation systems, in virtue of their ability to track the steps a student takes en route to a solution, are able to collect a tremendous amount of information on student problem solving strategies for later analysis.

Dynamic evolution driven by data

Perhaps the most significant design principle we have discovered is the need to make courses easily revisable. Within the EPGY software, every time a student is asked a question, his or her computer keeps track of a number of facts about the interaction, including how long it took him or her to answer that question, whether the answer was correct or not, and if incorrect what the answer was. This information is collected at a central location. By keeping track on which parts of the course students do poorly in and isolating the sources of difficulty, we are able to refine the course by adding new material designed to preclude these problems. If we find that a statistically significant number of students are getting a certain problem incorrect with the same incorrect answer, we can determine the source of the students' confusion and add appropriate remediation material to explain to these students exactly what they have done wrong. This process of data collection, analysis and refinement enables us to pinpoint those locations of the course which need the most revision and to therefore spend development time efficiently—perhaps the single biggest gain for a project with limited resources.