19. The Many Faces of a Computational Medium: Teaching the Mathematics of Motion*

Andrea A. diSessa

Graduate School of Education, University of California at Berkeley
Berkeley, CA 94720 USA

Abstract. This chapter reviews the development and deployment of two editions of a course on the mathematics of motion. The course was based on the premise that everyone involved—students, teachers and researchers—should develop a flexible competence with a general, programmable computational medium, Boxer. We illustrate the many ways that Boxer was used in the course, from microworlds and flexible tools to tutorials, from a compact and precise notation in which to define and use fundamental concepts, to providing the basis for extended and thoroughly personalized independent projects. A computational medium has many attractive qualities that help foster a gradual but effective shift in classroom practices to new ones that support more effective and more enjoyable learning of important mathematical and scientific ideas.

19.1 Introduction

Literacy, in the conventional sense of understanding and being able to produce written language, pervades our educational system deeply. It is, indeed, difficult to imagine what things would be like if literacy were not a taken-for-granted part of the competence of teachers and students. Of course, students and teachers learn and instruct the humanities by reading and writing, but things are really little different in mathematics and science. Textbooks usually dominate every phase of "ingestion," from organizing what is called a lesson, to being the direct focus for students (reading the text) on their first pass at learning the subject. On the second pass, problem solving, what fits nicely into a few sentences determines units and focus.

*This paper is based on an earlier version presented at the sixth international conference on Technology in Mathematics Teaching, Birmingham, England, September, 1993. The work described was supported, in part, by the National Science Foundation, grant numbers NSF-MDR 88-50363 and NSF-RED-92-52725. The opinions expressed are those of the author and not necessarily those of the NSF.

And modest technical extensions of language literacy (but still within easy reach of paper and pencil technology), algebra, graph drawing and the like, fill out the rest of the externalized thinking props that both constitute and develop understanding.

It seems inconceivable that the properties of such a fundamental part of thinking and learning do not, in some important measure, determine what is thought and how it is learned. One doesn't need the subtlety of a Whorfian hypothesis to see how children shut out of conventional discourse forms suffer in school.

Complementary possibilities constitute the most exciting prospects for education in a century or more. That is, by extending linear language into the multiply connected, dynamic, richly textured graphical and interactive forms allowed by computers we may fundamentally extend the material bases for thinking and learning, and with them the whole practice of education.

To oversimplify, there are two contrasting conceptions of a computationally enhanced literacy. The dominant one today I would describe as a trivial literacy. In this view, experts and software designers will supply the general populace with highly tuned and elegant tools and other pieces of software that we will learn to use in the niches for which they were intended—symbol manipulators, graphers, simulations and simulation tool kits, and the like.

In contrast, a deep computational literacy offers one crucial additional resource—in-principle access for everyone to the creation and modification of the dynamic and interactive characteristics of the medium, the very characteristics that define the medium as an extension and improvement of text in the first place. In metaphorical terms, reading without writing is only half a literacy. Deep computational literacy means "writing" in addition to "reading," creating as well as using.

Most would call this additional resource and set of activities programming. We prefer not to prejudice the form of access to creation and modification of dynamic and interactive structure by using a word loaded with assumptions about difficulty and social patterns (e.g., "programming is hard, and programmers do it for you"). Instead, we prefer to refer to the encompassing system that has full programmability—an intended basis for a new literacy—as a *computational medium.*

The issues concerning whether the development of a deep computational literacy is possible and desirable are complex. We make three main points about these here.

Nontrivial tool; nontrivial skills. First, powerful intellectual tools are naturally nontrivial to learn. Trivial literacy assumes that complete intuitive transparency and no learning curve characterize the best software. In contrast, the "no free lunch" correction to this prejudice against learning is that transparent power is very likely to have no effect on the individual, except in his/her local ability to accomplish. If our goals are educationally to transform individuals, then we should expect a long development that, in the end, results in generalizable intellectual power rather than just the ability to do something better by using a tool crafted for that particular purpose. Long development with powerful payoff ought to be familiar from traditional literacy. In the body of this paper, we point out many cases where student and teacher accomplishments built on substantial prior accomplishments.

Exponential cumulativity. The second point is directly related. A computational medium provides exponential rather than linear cumulativity. That is, a new competence with the medium enhances any context in which that competence might be applied. In contrast, for example, learning a highly tuned and targeted application means learning one more skill or capability. This is fine and appropriate. But when an expanded cumulativity is possible, it may be the better option.

"Small dollops," ownership and deep appropriation. Third, allowing creative access to the fundamental capabilities of the medium provides for the organic evolution of practice. Dropping dollops of software into teachers' and students' lives has serious appropriation problems, even if the software is excellent. Once again, "software dollops" provide only linear cumulativity. Furthermore, a piece of software typically obliges users to learn a new practice around it. Everyone knows that excellent software can be "used wrong," and teachers always struggle with fitting "a piece at a time" in with the rest of what they do.

A computational medium alleviates these problems. First, the "dollops" added may be quite small. Within the context of a computational medium that provides many capabilities "for free," each application may need to add only a few specific capabilities to be useful, as opposed to a complete self-contained world. That means new software is sometimes easy to develop—teachers or students may get into the development act. The ability to modify, hence adapt to local circumstances, is also a strength of a computational medium. Even "big dollops" of software are modifiable and extendible to the point that they may truly serve the community's sense of its own needs and personality. In short, a computational medium may change the grain size of "new software," the sense of connectedness with prior classroom practices and other work, and the basic ownership connection felt toward the software.

This chapter takes a concrete approach toward arguing the reality and importance of the above considerations. We examine the use of our computational medium, Boxer, in two editions of a course on the mathematics of motion. Basically, we simply list many of the forms of software that we, teachers and students developed as part of these two classes. Some of these forms (microworlds, tutorials) will be very familiar. Even for these, however, patterns in their development and use imply fundamental shifts in the practice of teaching and learning with a computational medium compared with conventional software. As I have emphasized elsewhere (diSessa, 1989) the surrounding practices and social relations for a piece of software are essentially always much better indicators of its effectiveness—and even of what it *is*—than any description of the software itself. Some forms of software we describe are less familiar (e.g., interactive representational forms) and more specific to the use of a computational medium. In these cases also, the underlying patterns of use are the critical issues.

There are many things this review will not cover. Each piece and type of software has many important properties—not the least of which is the learning theory behind them—that we cannot describe here. When possible we refer readers to more extensive descriptions. Neither will we be able to give an accountable description of the success in learning that these courses and pieces of software fostered. Instead, we need to rely on face value (which some of our recounting nonetheless unambiguously suggests) in order to concentrate on the underlying patterns of development and use. Those patterns are the best indications that the form of learning really has been substantially altered by the presence of a computational medium.

19.2 Overview of the Classes

Both classes we describe aimed at teaching kinematics, the mathematical description of motion. Among the topics covered were: concepts of instantaneous velocity and acceleration in one and two dimensions; relative motion, frames of reference and compositions of motion; vectors and vector forms of velocity and acceleration; graphing (e.g., position, velocity and acceleration graphs and relations among them); and at least qualitative versions of the basic idea of integration and differentiation. Although for the most part we tried to avoid causal aspects of motion (dynamics), both courses ended with versions of Newton's second law, which specifies the relations among force, mass and motion.

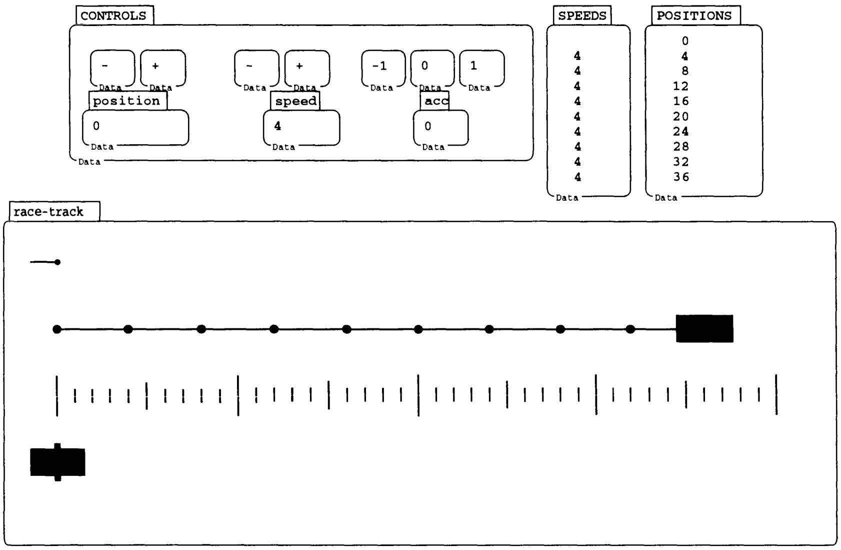
The first edition of the course involved eight sixth grade students, four male and four female, four days a week during a full academic year. Their school was academically-oriented and independent, located in Oakland, California. The students were bright and articulate, but not by any means prodigious.

The second edition took place during fifteen weeks in an independent high school in San Francisco. There were six males and two females. Like the sixth grade class, the students were volunteers, but unlike the sixth grade selection, these students skewed toward the low achievement end of the scale at their school. This course was greatly compressed compared to the sixth grade edition. Indeed, only ten of the fifteen weeks were spent on kinematics; the rest were spent on learning Boxer. Most of the students in both classes had not programmed before, and none of them had encountered Boxer.

Among the orientations that characterize these classes, apart from the use of Boxer, were an emphasis on learning through collaborative discussions and extended individual or group projects.

19.3 Microworlds

The ideal microworld should have the following properties: (1) It ought to have a fairly simple interactional form students can easily learn. (2) It ought to provide a wide range of self-motivated activities. (3) It ought reliably to bring students into contact with fundamental mathematical or scientific ideas.



```
SPEEDS   POSITIONS
            0
  4         4
  4         8
  4        12
  4        16
  4        20
  4        24
  4        28
  4        32
  4        36
```

Figure 1. In Number-speed two turtles are programmed to race by generating lists of speeds and positions. Here, the first turtle is programmed with initial position 0, speed 4, acceleration 0.

Figure 1 shows a main part of a simple microworld called Number-speed designed by Steve Adams, which was used about one-third the way through both courses. The idea is that a list representation for speeds makes an excellent approach to understanding some of the essentials of position, speed and acceleration. Students exercise the ideas (1) that speed determines a local property of motion, (2) that speed provides the unit time increment to position, (3) that, in complementary manner, differences in position determine speed, and (4) that the same set of relations (2 and 3) holds between acceleration and speed.

The main activity in the microworld was "turtle racing." Students programmed number lists by setting initial speed, position and acceleration. Acceleration was limited to plus or minus 1 and 0. One could then inspect the lists of positions and speeds and see the motion thus determined. A series of graded challenges were included. For example, students had to describe particular motions to their partners in words, and their partner then had to program them. The most complex set of challenges involved markers placed on the turtle racetrack at which point one or the other of two programmed turtles had to be ahead.

The presentation of this microworld is mostly entirely generic Boxer structure. It uses unadorned, basic structures with which every user is familiar. The number lists are simply data boxes (variables) that most students encounter in their first few sessions with Boxer. Data boxes may be edited directly at any time, or they may be changed by a program, which changes are immediately visible to users. Developers of materials using Boxer thus may use data boxes as user interface input or output devices, with no special IO statements or programming. Many times, IO and interface programming constitutes a principal effort in developing computer materials.

Using generic structure has one disadvantage and three critical advantages for students. The disadvantage is straightforward, a prototypical example of issue 1: nontrivial tools; nontrivial skills. Students need to be familiar with basic structures in Boxer in order that the open use of these won't constitute a serious block to getting things done. This is not a straightforward matter for designers of materials because Boxer, like any complex system, has both simple and more difficult to master components. So developing skills in using the simpler components effectively and generally anticipating the level of competence of students is critical.[1]

The three advantages of using generic Boxer components in microworlds all have to do with various manifestations of cumulativity, issue 2. First, using generic Boxer provides immediate familiarity based on prior experience. Students sit down knowing things that are possible and things they are likely to be asked to do. In our motion courses by and large we felt we were close to the ideal state—that students don't need to learn anything technically new (e.g., what is the nature of what they see on the computer screen; how should we interact with it?) in order to start engaging a new microworld conceptually. Number-speed was an example.[2] Success here was in some measure due to our accumulated experience as designers of Boxer materials. In contrast, a very early course using Boxer flirted with disaster because we seriously overestimated the ability of students to construct all their own tools from scratch. (See diSessa, 1993.)

The second advantage is the flexibility that cumulative skill allows. When students are presented with systems built from familiar parts, they have unusual

[1] A side effect of using generic Boxer structures in microworlds is that it is more difficult for people who don't know Boxer to get a sense for how easy or difficult the microworld may be to use, and even what things are possible with it.

[2] This is a design as well as a systems issue. It is as easy to add bells and whistles to microworlds in Boxer as with other systems. However, we steadfastly avoided doing this for these courses.

access to inner workings, whether to see how microworlds work or to modify or extend them. In the case of Number-speed we saw a stunning example of this, which we facetiously dubbed "learning by cheating" (Adams and diSessa, 1991). A pair of sixth grade students was working on the last challenge in the microworld curriculum, which involved three changes of lead in a turtle race. This is an impossible challenge (with constant acceleration)! These students came to understand the impossibility of the challenge by hand editing the speed list (ordinarily automatically generated by setting initial speed and acceleration) so that there were three changes of lead. Then they developed an argument that the pattern needed could not have been produced within the constraints of the actual program. We had many other examples of students productively using microworlds in ways we did not anticipate, some described below. Many of these were attributable to the fact that they had direct access to the "innards" of the microworlds.

The third advantage to students of using generic structure, like Boxer lists of numbers in Number-speed, is the cumulativity that builds from any powerful representational system. Lists of numbers are general, useful, and effective representations both computationally and intellectually. Students will have many contexts in which the representation actually does work for them. Boxer number lists were generated and used in many circumstances by students other than in Number-speed. One sixth grade student, in fact, did an independent project on "calculus" by implementing generic list processing programs that "differentiated" and "integrated" functions, represented by number lists.

Number-speed is an especially nice example of representational cumulativity because conceptual aspects of its use are so tightly intertwined with technical aspects. In the same way that by learning about graphing, one also learns about functions, number lists are powerful computational constructs for thinking about motion. We used them consistently in many circumstances, for example, as a mediating conceptual representation that allowed students more easily to translate between, say, velocity and acceleration.

In one episode, sixth grade students were discussing the position graph that would result from a velocity that linearly increased to a point and then remained constant. The students knew the first part of the graph would be concave upward, and the second part would be a slanted but straight line. The students debated whether there would be a "corner" where these portions joined, or would they join smoothly? (See Figure 2a.) One girl settled the issue, arguing that "the amount it (position) increases each time increases to a certain amount and then stays constant." The graph should slope more and more, and then continue at its highest slope. We interpret this as an implicit reference to a speed list that increments positions. It is as if she constructed a speed list such as in Figure 2b in her mind corresponding to the given velocity graph, then read out the implications of each speed for the position list and graph.

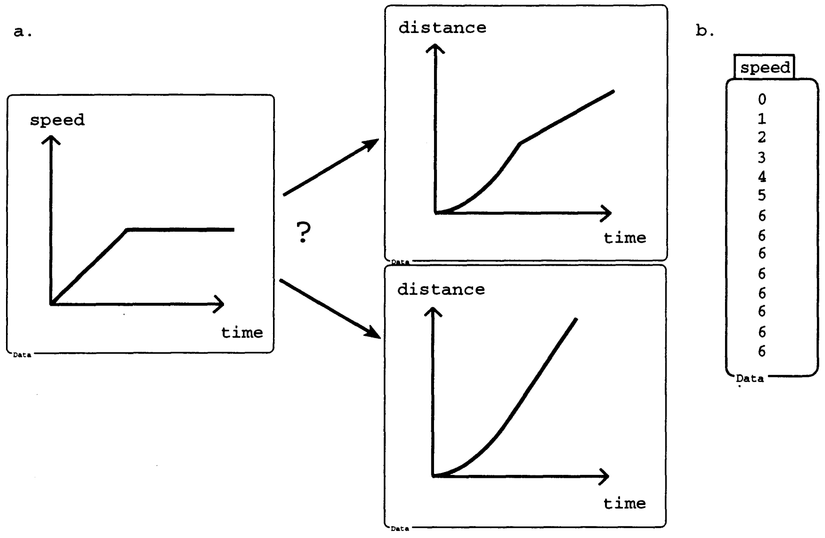


Figure 2. Generating a position graph from a velocity graph. (a) The question. (b) Number lists provide a tool to resolve the issue.

Figure 3 shows a relative motion microworld called Elmira. The large cross represents a frame of reference in which the small circular ball moves. One specifies a motion for the frame of reference and one for the ball. These motions are independently executable. We asked students to predict the combined motion that the ball exhibits with respect to the fixed background when it and its frame of reference are simultaneously moving. Students can slow the motion, "single step" it, or use a number of analytical tools such as showing the path of the ball in the moving frame of reference, showing the path of the moving frame of reference, or drawing the path of the ball on the fixed background frame of reference.

Elmira took about a day of programming initially to implement. Subsequently, implementing redesigned and new features based on formative trials took about another two days of programming. This is not unusual with Boxer. Many of our microworlds were implemented in only a few days. But it compares strikingly to the man-years typical of educational software development. The main reasons for reduced development time are the powerful general structures (e.g., for text editing and dynamic graphics) that are immediately useful for developers, teachers and students in Boxer. As a result of "quick prototyping," most of the time spent on a microworld can be on educational design, not coding.

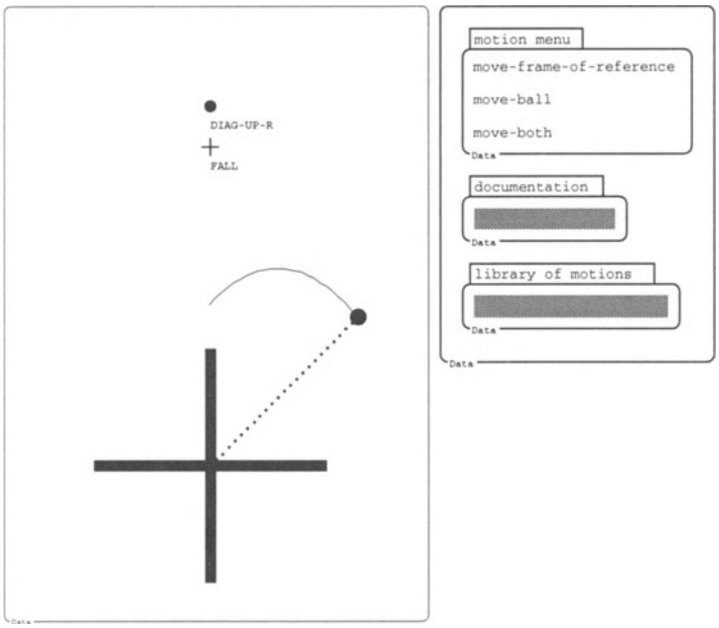


Figure 3. The cross is a frame of reference (here it is "falling"). The ball is moving diagonally upward (dotted path) in the frame of reference, resulting in a net parabolic path.

One of the tricks that makes quick development possible in Boxer is that we can rely on students' knowledge to fill in where otherwise much development programming is needed. Instead of elaborate error trapping, for example, errors are not unusual events for students who have programmed a computational medium. Most are handled as a minor annoyance, without interfering with the main intellectual work of the microworld. This observation is in line with the discussion on uses of generic Boxer structure in the Number-speed microworld. In particular, easing the developer's (or teacher's) burden relies on the cumulative power of students' learning Boxer (issue 2), and it is subject to the "nontrivial skills" caveat (issue 1).

Quick and easy development of materials is an important aspect of the organic growth of teaching practice (issue 3). We introduced Elmira to the teacher of the sixth grade class literally the day before she was going to use it. Although we felt we had a good design, the teacher had several strong reactions. First, she wanted to rearrange and simplify the menus for students. Second, she wanted us to remove all documentation as she considered it a distraction and complication rather than a help for her students. Finally, she wanted to modify and rearrange the sequence of problems. We made all these changes and the "new" microworld was used successfully the following day.[3]

[3] The teacher was a reasonably competent Boxer user, and we have no doubt she could have made the changes herself. However, it was expedient for us to save her the trouble.

Whether or not the teacher was correct in her redesign is beside the point. What she did was substantially to make the microworld her own. Her own style and sensibilities concerning her students could easily be established. In general, we see teachers as serious collaborative designers of Boxer materials, not only in initial design but also in the field. It is important that we can supply easily modified materials, and that what teachers learn in changing one microworld is applicable to others (cumulativity). We will provide other examples of the possibility and power of teacher and student modifiability below.

By now, it should be unnecessary to note how important it was that this teacher had a sense for what was easily possible in Boxer (nontrivial tools). Among other things, our relationship with her could not have been sustained if she constantly suggested impossible-to-achieve changes, e.g., if she wanted boxes to behave like windows. Teachers without her accumulated competence and confidence could not engage in the ownership-through-redesign process.

Here we note just one feature of Elmira that fostered student modifiability via exponential cumulativity. The motion library that contains all available motions for the microworld is completely student accessible and extendible. The representation of the motions is number lists, in consonance with Number-speed and the curriculum more broadly. Because students may use this microworld before number lists are familiar, we included a translator in the library that could turn a more familiar representation of motion, a turtle program (move forward, turn right, ...) into number lists. In our experience students like to play with unusual motions like "reverse gravity" (the frame of reference "falling up") and irregular motions. Because of Boxer's features, including the general knowledge students have about perusing and editing Boxer structure (in this case, the library) the translator took only about 15 minutes to implement.

Elmira is among the best studied of the microworlds we made for these courses (diSessa, 1989; Metz and Hammer, 1993). Studies show that students can effectively learn Elmira using dynamic spatial intuitions rather than formal constructs like equations and functions. This helps validate a central assumption behind the development of computational media—that new graphical, dynamic and interactive structures can genuinely develop forms of intelligence that standard literacy is inadequate to reach.

Figure 4 shows a microworld that we introduced to students about the same time as Elmira. It is called Tinker and contains magical tinker-toy rods, each of which has a characteristic motion. Some are fixed in length and direction. Some grow at a constant rate. Some turn perpetually in a circle. Students can hook a series of such rods, end to end, and watch the resulting complex motion of the end of the final rod. This microworld concerns the composition of motions, and it provides a very concrete image of the vector decomposition of motions. We use Tinker as a modeling kit in which students explore building complex motions out of simpler ones. For example, the composition of a constantly growing tinker rod and an "accelerating downward" rod produces every form of gravitational toss—parabolas of all sorts, a simple toss straight up and fall back down, or just a uniform "drop from rest." Composing circles appropriately leads to sinusoids (or more generally, epicycles), and composing sinusoids appropriately leads to circles or ellipses.

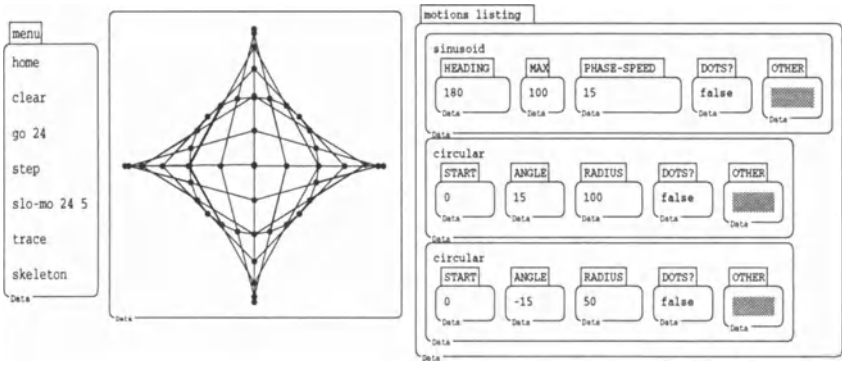


Figure 4. The trace of a sequence of "tinker rods," having the motions specified by the three boxes in "motions listing." "Trace" (show the path of the end tinker) and "Skeleton" (leave images of all tinker positions) are turned on.

Tinker, even more than Number-speed or Elmira, relies on making generic Boxer structure available to students. The list of tinkers that students generate and modify is a sequence of boxes in a box, each of which contains the computationally active parameters of the program that runs the microworld. Thus, in a real sense, students are modifying the code of the microworld directly rather than an interface to it. While we provided some shortcut means to make the most usual changes to the system of tinkers, students could and did reach directly into the "tinker code" to make more exotic changes. Sometimes, students avoided the shortcuts we provided because, presumably, directly editing the box representation of tinkers was completely understandable to them.

Tinker illustrates another of our design heuristics in constructing microworlds. We added some features that were designed to allow students to explore the aesthetics of compound motion, as well as its mathematics. Figure 4 shows a sample product. This again extends "school stuff" into the personally meaningful parts of students' lives. A computational medium should blur the boundaries between forms of activity, school versus "having fun," allowing deeper appropriation of the medium and ideas expressed in it. Designing microworlds for aesthetics, as well as mathematics, reminds us that appropriation has to do not only with technical features, but also with the attitudes, intentions and strategies of those using it.

We did, of course, develop more microworlds for this course than we have time to describe. So we close this section with a few anecdotes about some of the central features of microworlds constructed in a computational medium (rather than as stand-alone applications). These also have to do with issue 3, ownership and deep appropriation, via personalizability and extendibility.

The "learning by cheating" episode above was a dramatic, but not really unusual occurrence. In Number-speed itself, for example, all the students in the sixth grade class spontaneously (with each others' help) added a command to the menu in the very first exercise of the microworld. We had wanted to introduce the idea of generating number lists in the abstract. But the students discovered how to activate motion based on the number lists, and all decided it was fun or useful to do so. We were happy to let them set the pace in this way as our own judgments about how to introduce exercises were tentative. In many other microworlds, as well, students made small or large changes. One high school girl added some missing features to Elmira that she felt she wanted.

One very telling incident occurred when I visited the sixth grade class to observe. A boy and girl who were using a microworld I had developed were evidently a little bored[4] and diverted themselves by selecting large chunks of the microworld, deleting them and then pasting them back. I asked them if they weren't worried about doing that. The girl replied she was not because the microworld was really simple, and she could "put it back" easily. Whether or not she could do this without having to find a fresh, undamaged version is, again, beside the point. These children evidently felt they owned and could do with Boxer microworlds as they liked.

19.4 Tutorials and Exercises

Although tutorials and "computer exercises" are not characteristic of our pedagogical style, we did develop and use a number of these. Tutorials are extraordinarily easy to develop and modify in Boxer. Text editing is a capacity that one uses from the first encounter. Boxes inside boxes take only a keystroke per box to construct, and these provide a nice hierarchical form to help students get overviews on a clear conceptual organization of the material. Expanding and shrinking boxes provide means for controlling complexity for the creator and user of a tutorial.

Because a computational medium is fully integrated with programming, any dynamic or interactive chunks may be included in a tutorial. These may only be "moving pictures," or complete simulations. General tools that have been built for other purposes can also trivially be imported into or used to fabricate pieces of a tutorial. A graphing tool can simply be cut and pasted into the midst of a tutorial, if it would be useful for a student to use at that point. This is a material form of cumulativity.

[4]Yes, our students were not always at-the-edge-of-their-seats enthusiastic about everything we asked them to do!

An example that we shall describe in more detail later are vectors that a graduate student programmed as an extension to Boxer. Vectors are "graphics boxes" than one can make with a keystroke. They contain an arrow, representing the vector, which can be modified in direction or length by clicking and dragging with the mouse. Any graphics box, including vectors, may be "flipped" to reveal non-graphical structure. In this case, we had coordinate numbers shown (and modifiable) on the flip side. The critical feature of Boxer vectors is that they are completely computationally active. One can execute "add V and W" and see the resultant vector; one can tell a Boxer graphical object to move along a particular vector.

Writing a vector tutorial in Boxer involved writing some text and inserting vectors at appropriate places. A line or two of code can generate illustrations of vector properties, or dynamic images of vectors controlling motion.

From the student perspective, Boxer tutorials are open and flexible, hence personalizable and appropriable. Students can write notes for themselves or respond in writing to teacher's questions at any point. One effective feedback strategy that some Boxer teachers use (see Noss, this volume) is to leave notes (or even code-fragment suggestions) for the student to discover on returning to his/her work. A nice example of Boxer's flexibility in support of student initiative, in fact, occurred in the vector tutorial. In the section on properties of vectors, one pair of high school students decided on their own to try out what happened in shuffling around three vectors, rather than the two that were used in the tutorial to illustrate commutativity. It was an easy matter for them to "change the code" in the tutorial.

Tutorials and exercises in Boxer are frequently the place where students learn not only a concept, but how to implement that concept in a program. Vectors also provide ready examples of this. Part of the vector tutorial had students fill in a line or two of code using vectors to create a particular motion. Cumulativity with a computational medium, as we pointed out with Number-speed, can be both conceptual and practical. For example, programming with vectors became the basis for many individual student projects.

Another example of an exercise we constructed for students was a nearly complete program, assembled in a few minutes and implemented with vectors, of a ship moving on the surface of a moving ocean. The student exercise was to fill in the single line of code that caused the ship to move appropriately, give its motion and the ocean's motion expressed as vectors. The exercise worked well to connect relative motion to vector addition.

19.5 Flexible Modeling Language

Programming, if it is a part of general student literacy, is an excellent general modeling language. In a sense, it is a universal modeling language and as such may provide exceptional cumulativity, subject to caveats associated with other general but highly nontrivial tools. We made a couple of uses of programming in this mode in our motion course. Indeed, the very first exercise in these motion classes was for students to create simulations of a number of familiar motions, like a book being dropped or sliding to rest after a shove on a table, or a dot painted on a rolling tire. For students, this was an introduction to the complexity (or simplicity) of analytically describing real-world motions. For us, it produced some important insights on how our students began thinking about motion. Only one pair of sixth grade students initially thought it was important to display the speeding up of an object in a fall. And they chose to depict that speeding up with an object that actually slowed down during its fall. It slowed down because they chose to depict "more speed" with more dots drawn by the moving object! (See Figure 5.) This is an unconventional but cogent representational form that is actually common among children and less technically sophisticated adults. The representation would be easy for a teacher to misinterpret. Windows into conceptual and representational competence of students, such as offered here by a simple programming/modeling task, are important in any constructivist instruction. This example also shows that modeling need not be an esoteric activity carried out with hard-to-learn specialized tools.

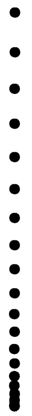


Figure 5. A student representation of an object gaining speed in falling. A conventional representation would depict equal time intervals, leading to increasing dot spacing.

19.6 Compact and Formal Representations

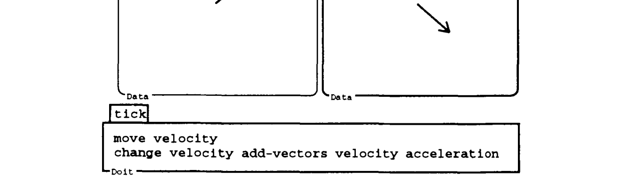
Perhaps the most controversial aspect of the way that we taught motion with a computational medium is that we used programming essentially to replace algebra as the formal representational language in which to express basic definitions and fundamental laws. Note the form of the argument here. Everyone knows algebra is difficult (a nontrivial tool). Yet few dispute its cumulative power and dismiss it instructionally just because it is difficult. We are arguing that the same kind of considerations and balanced judgment must be made for programming. Programming may even have greater cumulativity in being useful in many other ways than as a formal representation, and it arguably requires less instruction than algebra, at least as used in these courses.

There are many deep epistemological and pedagogical issues concerning using programming as a compact, formal representation into which we cannot enter here. (See, for example, Sherin, diSessa, and Hammer, 1992.) We argue here in a very abbreviated form that using programming in this way is a good thing to do.

(1) Motion is an interesting topic that is demonstrably teachable to much younger children than those to whom it is usually taught. Sixth grade children don't know algebra, but they can easily learn the programming necessary to move things around. If there is any general analytic precision that can be brought to motion for these children, it is through programming.

(2) Programming representations of motion are fundamentally more interesting than, say, algebra. Programming makes things actually move and interact, and it allows treating more complex types of motion than solving equations.

(3) Programming involves discrete, one-step-at-a-time models of motion that are important, psychologically tractable intermediate models of continuous processes.



```
move velocity
change velocity velocity + acceleration
```

```
move velocity
change velocity add-vectors velocity acceleration
```

Figure 6. The "tick model" in one and two-dimensional forms.

The core of our use of programming as a formal language is the "tick model." Figure 6 shows scalar (one-dimensional) and vector versions of the tick model. Two variables, v and a, represent velocity and acceleration. At each tick of the clock, the procedure *tick* is executed in which: (a) an object moves a distance v

(thus defining velocity as a small chunk of motion); and (b) velocity gets changed by being incremented by the acceleration.

We taught the tick model in tutorials for both one and two dimensional (vector) motion. We used it again and again as a reference representation for thinking about motion. And we encouraged students to use it, as appropriate, in their own projects.

19.7 A Medium for Collaborative Design

In the high school course we essentially asked students collectively to design a version of the tick model that expresses important aspects of Newton's second law—in the absence of forces motion is a constant vector, and forces act by adding vectorially to existing velocity. This collaborative design was (with qualifications) a success and is documented in detail in Sherin, diSessa and Hammer (1993) and diSessa (in press). Our discussion here must be brief. Above all, it is completely evident from our review of the videos of this group effort that it was extraordinarily dependent on many aspects of the course that came before. That is, cumulativity was a critical feature in its success. For one, the use of vectors in programs was crucial in bringing clarity and forcing precision in the discussion. For another, it was clear that both the programming the students had done (in, for example, the ocean motion exercise—composition of motions) and also the conceptual understanding they had gotten from using programs were instrumental parts of this effort. The course had built an appropriate conceptual cumulativity, and programming was a critically intermixed representational cumulativity.

While considering the collaborative aspects of "designing Newton's laws" is a good time to emphasize that media of any sort can make only limited contributions to educational success. Boxer did, as suggested above, help students understand each other in appropriately precise terms. See also diSessa (1993) for further accounts of sometimes remarkable collaborations and reasons we believe Boxer supports these. However, collaboration is a complex and delicate affair. No computer system can "fix" cultural gaps or similar impediments. Similarly, the dedicated and systematic effort of our teachers to encourage a collaborative spirit and strategies beyond question deserves recognition.

19.8 Tools

The principles behind tool building and use are quite similar to those behind literacies. Tools creation represents material cumulativity in embodying the intelligence of the designer. Tools are frequently nontrivial to learn, but warrant learning for their power and the personal expressiveness they liberate. A computational medium is a wonderful context in which to develop and share tools. In fact, one of my favorite images of educational practice based on computational media is as tool-building and sharing for learning. Vectors, again, can serve as an excellent example. Graduate student Bruce Sherin, who developed Boxer vectors, facilitated the development of many tutorials and microworlds. We mention two more here, and an additional one later.

We decided that relative motion was an excellent introduction to two dimensional kinematics, and so we developed a simple game microworld to start students off.[5] "Cheeser" has a mouse running across the surface of a table, trying to reach and nibble a piece of cheese hanging from a thread down almost to the tabletop. The table is also moving according to some regular or irregular program, so the mouse must react to, counter and compensate for any motion of the table. Students "played mouse" by manipulating a vector representing his running speed and direction in real time.

A simpler microworld had students directly manipulating the vector velocity or acceleration of an object to accomplish specific tasks, like orbiting a planet or docking a spacecraft with a moving mother ship. Both of these were easy exercises for us to write based on having interactive vectors as an extension of Boxer programming.

Another tool that we introduced to the course was a simple graphing utility. However, for the most part, we had students, if they wanted to, write their own graphing programs. This was within their programming competence and, we judged, both educational and helpful in preserving their ownership over the tools they used.

19.9 "MBL" Style Activities

Adding inexpensive interfaces to external sensors (microcomputer-based laboratories) allows an important style of software that is easily accommodated in a computational medium. Instead of complex smoothing algorithms and automatic graphing, we chose to give students access to the raw data from a motion sensor. We felt the flexibility and reality of dealing with real data, errors and all, outweighed the negative consequences of such a low-level interface. One exercise, which proved surprisingly difficult, was simply to have a graphical object trace a (scaled version of) the motion of an object that was being moved in front of a distance detector.

Another "real world" activity involved taking strobe photographs showing multiple exposures of tossed and dropped objects. We had worked up to this experiment by having students debate various issues, such as how an object speeds up in falling, or whether the horizontal component of a tossed object's motion decays. Some sixth grade and high school students (correctly) believed the velocity of a falling object would be incremented at regular intervals by a constant amount. Some thought it would be multiplied by a constant factor. One sixth grade girl thought the

[5] It is no accident that relative motion and compositions of motions defined so many student activities. We decided these are conceptually foundational, so emphasized them.

motion would have a constant jerk[6] (third derivative—which she had encountered herself by extending the tick model, adding a variable that incremented acceleration!). Most students believed the horizontal component of motion of a tossed object fades, either spontaneously or as a result of the "interference" of gravity.

The days leading up to the strobe experiment in the high school class had not gone as well as expected. So the night before data analysis we built a tool that would simplify the exercise for them. Basically, students had to adjust a series of vectors (accelerations) to get a graphical object to follow the scanned-in images of dropped and tossed objects. Once again, a student had a better idea and, with help, he was able to extend our tool to deal with horizontal and vertical components separately. Adaptability and flexibility are key contributions of a computational medium toward personal and group ownership, toward relevance and fit of software to context of use and, overall, toward appropriability of both software and new educational practices.

19.10 Projects and Personal Work

Students generally liked personal projects and classroom discussions (such as "designing Newton's second law," described above) best in the courses. We provided several occasions to accommodate projects. Of course, these were of variable interest to us as educators with respect to developing motion concepts. But the personal involvement that projects brought was almost always gratifying. Projects are an important context in which computational media provide critical resources to help students take personal ownership over ideas and software, fostering a deep appropriation of both.

Early in the sixth grade class, we invited students to contribute exhibits to a "speed museum," depicting and illustrating aspects of speed they had learned so far. One girl got enthusiastically involved with a "bouncing babies" game, where Adorable and Cutie were babies tossed out the window of a burning hospital. The player had to move a net under them to save them or else they bounced on the pavement. This student demanded extra time after school to work on her project, as did several other students.

Two other girls worked on a scale model simulation of "carelessly speeding José Canseco" (referring to a famous baseball player who got into trouble repeatedly for speeding). "Careless José" runs down "Innocent Sue" with his Corvette. It turned out to be conceptually challenging to get the speeds of Sue's walking and José's driving to proper scale in the context of a depicted 200 foot long stretch of sidewalk. The students spontaneously used two different conventions to represent speed, both of the form "time to traverse a specific distance." The two conventions

[6]We informed the student of this conventional name after she had spontaneously explored the idea.

became a topic of conversation with the teacher, as well as how to convert these to more conventional representations of speed ("inverting fractions").

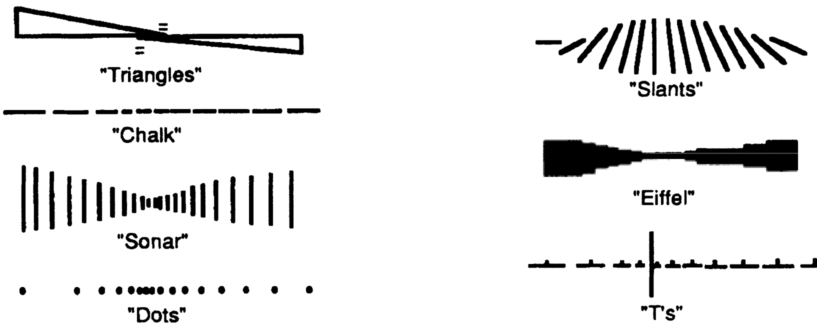


Figure 7. Representations of motion invented by sixth grade children. "Slants" uses slope to represent speed. "T's" uses dashes for speed, a vertical bar for time.

A boy in the class wrote an elaborately illustrated tutorial that showed and explained several motions in various different representational formats invented by the class, including graphing speed versus time. Figure 7 shows several of these representations (diSessa, Hammer, Sherin and Kolpakowski, 1991). Another produced a stunning graphing adventure game that presented multiple levels of challenges at identifying graphs that matched various motions encountered in an elaborate fantasy story, which was contributed mainly by another student. The adventure game had explanations of graphing conventions as well as playing instructions.

In the high school class several of the projects were motion games. One was a "lunar lander," where the player tries to land a space craft on the moon by aiming and firing its control thrusters. Another was a challenging game to dock a boat in swirling currents. Most computer projects used vectors.

One group of students did an interview study of misconceptions about motion among their colleagues and teachers. Like many cognitive researchers before them, they discovered that people seem to "know" many things about motion that are not true. The point of this last example is that we treat a computational medium as a ready resource to be used when appropriate, but not always!

One extremely interesting social phenomenon occurred at the elementary school in which we taught our motion class. Our collaborating teacher instigated a "library" in which students could place Boxer work of which they were proud. The library grew, run essentially by students. It contained mostly games; an excellent "Wheel of Fortune" (a popular television game) was a particular favorite. The library became a social networking device in which students saw what others were capable of, and from which they "stole" ideas and code for their own use. Some non-Boxer students were introduced to Boxer in this way, especially by means of a tutorial in the library that was written by one student. Many times neophytes joined Boxer-competent friends playing with their library creations after school and learned especially when spontaneous debugging or extending occurred.

In seeing the influence of a computational medium, it is especially interesting and important to remark on large and substantial student constructions. Some of the above projects fall into this category, and other descriptions of complex programming accomplishments can be found in diSessa (1993) and Ploger and Lay (1992). In contrast, we close this section with a "miniature" that represents an important complementary focus—how the medium finds its way into little corners of the ebb and flow of daily activities. During one of the lessons on relative motion at the high school, students were broken into groups for 15 or 20 minutes to consider three related aspects of what happens when an object is dropped from a moving support (Galileo's cannonball dropped from the mast of a moving ship). One group had to consider what this looks like (looking up at the dropped ball) in the moving frame of reference and reported their results with a simulation showing a fixed black disk that grew in size. The nice thing about the example is that there was no fuss or fanfare about it at all. The students considered a simulation to be a completely acceptable and unremarkable way to answer a question.

19.11 Helping the Developer/Researcher

A computational medium should be a flexible tool for everyone, not just for students. All the same issues of nontrivial tools and skills, of cumulativity, and of ownership, personal expression and appropriation apply to researchers and developers as well as students and teachers. We have described, above, some of the ways Boxer helped us quickly produce materials that effectively supported our educational goals to help students and teachers. Indeed, we feel it would have been completely unfeasible to develop a full-year computer based course with only three months development lead time, as we did, without Boxer. Here we briefly describe three rather different uses of Boxer that helped us do our own work efficiently.

In order to collect and analyze student data from using Number-speed, Steve Adams extended the microworld so that it saved data from all the trials students made in working on challenges. He also developed a tool for himself so that he could replay and review students' work while taking notes in Boxer on what they were doing.

I developed a much simpler, but still very useful Boxer environment in which to analyze protocols of students' work with Elmira. The environment had protocol transcriptions organized into chunks (boxes) according to the puzzle on which students were working. In addition, I had a series of thematic analysis boxes (e.g., student intuitive ideas, metacognitive reports and references to real-world phenomena) in which I collected notes on those topics. Most notes had links to the sections of the protocol where the issues arose, making it very easy for me or colleagues to review the evidence and surrounding context for any claims made in the thematic analysis. Links among the themes could also be represented.[7] We have since added the capability to Boxer to control a video tape recorder—for example, to request timer markings that correspond to a particular section of video and to locate any such segments in the tape by their timer markings.

Finally, in preparing the sixth grade class, we developed one central Boxer "database" that contained all of our work for the course. It contained all of our microworlds, both finished and in progress, our curricular plans and sequencing, lists of theoretical issues, plans for foci for data collection and even work assignment for individuals. One notable success of this data base was in helping new members of the group get a sense for all the things that were going on, who was working on what, how different aspects of the project related, and what were the important issues that concerned us.

19.12 Conclusion

We have described many of the ways a computational medium, Boxer, became involved in teaching two courses on the mathematics of motion. We developed microworlds for students, which combined experience with fundamental ideas with a flexibility that could encompass teacher and student initiative in conceptualizing and exercising those ideas. We developed tutorials and exercises for students that contained modifiable dynamic and interactive representations. Students used programming both as a general modeling tool to explore motion, but also as a formal and precise representation of basic definitions and constructs. They used it as a medium for collaborative design and as an expressive medium to enhance collaboration more generally. Developing general tools, like Boxer vectors, not only enhanced our ability easily to develop learning materials of all sorts, but, as well, these tools served students directly in many ways. For example, they served students as working contexts in which ideas (like vectors) did understandable work for them in getting objects to move in comprehensible ways. Tools and general programmability also allowed us easily to incorporate sensing apparatus and other real-world data and analysis into these courses. We emphasized how a computational medium provided excellent support for student independent work. Last, and probably least, we discussed how the medium also supported our work as developers of educational materials and activities and as learning researchers.

[7] In Boxer, such links are represented as "ports," which are views of any box from some other position in the box hierarchy. Ports allow complete access to the remote data, for example, allowing you to edit it, if need be, without having the problem of different versions of a segment (e.g,. "the original" and any quotations from it) getting "out of synch."

Through these examples we have tried to illustrate three general points about the use of a computational medium in instruction. First, while we readily admit learning a computational medium is not an easy task we hope that we have shown some of the intellectual and practical power students and teachers can achieve by gradually mastering an expressive new extension of externally-supported thinking. Perhaps the single most provocative claim implicit in these descriptions is that, with a computational medium, we have managed to teach some important mathematics of motion in an unusual but cogent form to students (sixth grade) long before they ideas are usually encountered.

The second point we have tried to illustrate is that learning with a computational medium is exponentially cumulative. Using programming and the general features of Boxer in a learning experience, say a tutorial or microworld, enhances students' abilities to use these in every future encounter with Boxer. A strong cumulativity operates materially, as well as intellectually. Any tool that is produced may trivially be included in any future production, or parts of it may be cannibalized and extended to very different circumstances.

Finally, and perhaps most importantly, we have illustrated how a computational medium can foster a more organic growth and change in the learning practices in classrooms. Tools and software can enter the community in smaller, more appropriable chunks. Every chunk, no matter what size, is open and modifiable so as to accept innovation, and with it, the local context, style and personality of the community and the individuals in it. The longest and most impressive line of appropriation discussed in this paper concerns extending Boxer to include graphically and computationally active vectors. Teachers and other developers can develop microworlds and tutorials, or other tools of analysis (e.g., for stroboscopic data) to their own taste with Boxer vectors. Students can (and did) modify and extend these new creations. Students also incorporated ideas, programming techniques and vectors themselves into extended, personally meaningful projects.

Acknowledgments

I thank Celia Hoyles and Richard Noss for comments on a draft of this chapter. Thanks are also due all the members of the Boxer Project who contributed to the development and running of our motion courses.

References

Adams, S. and diSessa, A. A. (1991), Learning by cheating: Children's inventive use of a Boxer microworld, *Journal of Mathematical Behavior*, 10/1, 79-89

diSessa, A. A. (1989), A child's science of motion: Overview and first results, in U. Leron and N. Krumholtz (eds.), *Proceedings of the Fourth International Conference for Logo and Mathematics Education*, 211-231, Haifa, Israel: Israeli Logo Center, Technion—Israel Institute of Technology

diSessa, A. A. (1990), Social niches for future software, in M. Gardner, J. Greeno, F. Reif, A. Schoenfeld, A. diSessa and E. Stage (eds.), *Toward a Scientific Practice of Science Education,* 301-322, Hillsdale, NJ: Lawrence Erlbaum

diSessa, A. A. (1993), Collaborating via Boxer, in P. Georgiadis, G. Gyftodimos, Y. Kotsanis, and C. Kynigos (eds.), *Logo-like Learning Environments: Reflection and Prospects*, Proceedings of the Fourth European Logo Conference, 351-357, Athens, Greece: Doukas School

diSessa, A. A. (1995), Designing Newton's laws: Patterns of social and representational feedback in a learning task, in R.-J. Beun, M. Baker, M. Reiner (eds.), *Dialogue and Instruction,* NATO ASI Series F, Vol. 142, Berlin: Springer-Verlag

diSessa, A. A., Hammer, D., Sherin, B. and Kolpakowski, T. (1991), Inventing graphing: Meta-representational expertise in children, *Journal of Mathematical Behavior*, 10/2, 117-160

Metz, K. E., and Hammer, D. M. (1993), Learning physics in a computer microworld: In what sense *world*? *Interactive Learning Environments*, 3/1, 55-76

Ploger, D. and Lay, Ed. (1992), The structure of programs and molecules, *Journal of Educational Computing Research*, 8/3, 347-364

Sherin, B., diSessa, A. A., and Hammer, D. (1992), Programming as a language for learning physics, Paper presented at the annual meeting of the American Educational Research Association

Sherin, B., diSessa, A. A., and Hammer, D. (1993), Dynaturtle revisited: Learning physics via collaborative design of a computer model, *Interactive Learning Environments*, 3/2, 91-118