WebLab!
A Universal and Interactive Teaching, Learning, and Laboratory Environment for the World Wide Web[1]

Christopher M. Boroni, Frances W. Goosey, Michael Grinder, Rockford J. Ross, Paul Wissenbach
Computer Science Department
Montana State University
Bozeman, MT 59717
ross@cs.montana.edu

Introduction

It's here. . .it's finally here! For those of us designing and/or using program, algorithm, and computer science concept animation systems, the one universal environment we have yearned for is finally available. That environment is the World Wide Web in concert with standard Web browsers, the Hypertext Markup Language (HTML), and Java. Now, educational software systems that have historically been tied to a particular type of computer and operating system can be ported to Java and made universally available on the World Wide Web. The potential impact of the alliances between this newly emerging technology, educational animation systems, and hypertext delivery mechanisms on standard college instruction—as well as distance learning, virtual universities, and even high school teaching—is profound.

The World Wide Web and its browsers are already finding many uses in computer science education. It is used for mail, news, local syllabi, research assignments, class notes, and even hypertextbooks. In this paper—in keeping with one of the focuses of SIGCSE '97 (papers with a futurist theme)—we look into the future at the next step in this evolving use of the Web for educational purposes, a step made possible by Java and Java-ready browsers, by presenting the impending role of the World Wide Web as a universal medium for the dissemination and use of computer science animation systems. A brief review of animation tools is given, followed by a short description of known work that is progressing in Web-based computer science animation systems. Then, our own work in program, algorithm, and concept animation underway at Montana State University is discussed in more detail. We close by presenting some formal evaluations of student learning in the presence of animation systems, followed by a short summary.

The Status of Computer Science Animation

A number of animation tools for studying computer science have been implemented and tested. These tools fall into three general categories: program animators, algorithm animators, and concept animators.

Program animation refers to the activity of illustrating on a computer monitor what is happening as a program is running by highlighting the portion of the code that is currently being executed, displaying changing variable contents, showing the effect of recursive calls, and so forth. In some ways, therefore, a program animator is reminiscent of an interactive debugger; in reality, though, a good program animator is much more oriented towards teaching and learning than debugging. Thus, a program animator—unlike program debuggers—will be extremely easy for a novice to use and will include unique features, such as time complexity measures, for learning about programming and experimenting with computer science in a laboratory setting. The best example of a program animator is the DynaLab system of Ross [2, 4].

Algorithm animation refers to the activity of visualizing in pictorial form the process behind an algorithm. So, for example, the animation of the binary search tree ADT would display the tree on a computer monitor and have options for the user to insert a new value, delete a value, and look up a value. In the case of an insert, the animation would show the new value descending the tree from the root—deciding at each interior node whether the value should proceed down the left or the right branch—until no branch existed, at which point the value would be inserted as a new node. No program code would actually be shown in this process, although some systems allow for both a "code view" and an "algorithm view." Current work in algorithm animation is being pursued by Brown [5], Stasko [6], and Naps [9] among others.

Concept animation refers to the activity of animating a particular concept from the science of computing, such as $\mathcal{NP}$-completeness, or the instruction fetch and execute cycle of a processor in the presence of interrupts.[2] Less work has

[1] Support for the project described here has come in part from the National Science Foundation, grant numbers DUE-9354655, DUE-9552411, and DUE-9455588; and a 1996–97 MONTS grant from Montana State University.

[2] An animation of the instruction fetch and execute cycle was done using

been done in this area. Rodger [11, 3] has developed animations of LL and LR parsing strategies, as well as deterministic and nondeterministic automata.

In all of the cases cited above, the animation tools described have been class tested and have demonstrated great potential. Yet, in spite of this, their actual classroom use has been limited to relatively few institutions. Why? The primary culprit has been *platform dependence*. All of these systems depend on a particular computer/operating system environment to run. Early versions often required X-windows with a certain version of Motif. Others ran only on Macintoshes. More recently, versions have been developed for PCs with Microsoft Windows. Regardless, the dependence on a particular platform has limited the use of these animation systems.

The future looks much different!

The Web and the Future of Computer Science Animation

Suppose that each of the systems described in the previous section were ported to Java (or some more stable language, such as Ada 95, for which compilers exist that produce J-code). Then these systems would run on any computer that supports Java-ready Web browsers, such as recent versions of Netscape and Internet Explorer. For all practical purposes, this would mean that these animators would be universally available! Not only that, but they could be used without even the hassle of local installation required of most current systems. Furthermore, students and instructors alike are so accustomed (or soon will be) to using Web browsers that the interfaces to these animation systems would require very little time—if any—to learn.

This fact certainly has not been lost on animation system developers. Every researcher mentioned earlier has begun to focus attention on the Web and Java as the platform for their animation systems. Rodger [12] has begun to develop animations of nondeterministic automata for the Web. Naps has embarked on a project to develop an algorithm animation environment with Java [8]. Brown and Najork have a prototype of an interactive textbook environment, complete with algorithm animations, for the Web [5]. Berman has described an effort to move the work of Stasko to the Web [1]. Ross is actively working on porting the DynaLab program animator to the Web while simultaneously extending DynaLab to include a concept animation component. In fact, there are many small, illustrative Java applets already available from a variety of sources that one occasionally stumbles upon while surfing the Web.

An Example—a Finite State Automaton Animator

At MSU, as noted earlier, we are in the process of porting DynaLab to the Web. In the process we are beginning to incorporate concept animations as well as the program animations of DynaLab. In order to test our ideas, a Java animator for deterministic finite state automata (fsa's) has been developed by the third author. A snapshot of an animation of an fsa is given in figure 1. (It can be tested firsthand at

`http://www.cs.montana.edu/~dynalab`

The animation continues to be modified and so will not appear exactly as shown in the figure.)

Using this fsa animator, a user can construct an arbitrary fsa, designating initial state, accept states, and transitions with labels, all with complete flexibility to alter the design on the fly as necessary. Then, arbitrary strings can be entered and run on the fsa either in continuous or step mode. The state shown in figure 1 with the enclosed shaded circle is the current state; the state with the double circle is an accept state. As the fsa runs, the small shaded circle moves along the correct arc to the next state, effectively showing state transitions. The position of the read/write head is indicated by the rectangle over the "c" in the string at the top of the panel containing the fsa. (Not seen in this figure is the effective use of color during this process.) Currently, a library of preconstructed fsa's is also being incorporated so that specific fsa's can be viewed as desired by all users of the animation.

Since the animation runs as a Java applet, the animation can be used in the PC labs utilized by our introductory students. It can also be used by advanced students in our theory of computing course in which X-terminals are the more common platform. Instructors can carry a notebook computer to a classroom and construct and project an fsa on a screen for general lecture purposes. And, of course, students can access the system on their own from any place and any computer providing access to the Web, allowing homework, self-paced laboratory exercises, and individual study to be based on the animator. The beauty of all of this is, of course, the complete freedom from worry about which platform a student or instructor may be using: the Web is universal.

At the time of this writing, the semester is just beginning, so there has been no opportunity to class test this particular animator. However, there are plans to use it in the laboratory-based, four-semester lower division (CS I–IV) course sequence at MSU [10]. The faculty member in charge of the graduate and undergraduate theory courses has also requested to use it this year.

More animations will be added as the year progresses. The goal is to develop a substantial repertory of animations from the theory of computing, targeting especially topics that typically give students the most difficulty, including the various automata, nondeterminism, grammars and languages, pumping lemmas, problem reductions, and—eventually—$\mathcal{NP}$-completeness. Keep watching the DynaLab Web site for developments!

Porting DynaLab to the Web

As noted earlier, the DynaLab program animator is also being ported to Java. Once this is accomplished, students and instructors everywhere will have access to a comprehensive,

---

the Polka system of Stasko by the last author of this paper and is being used in classes at MSU.

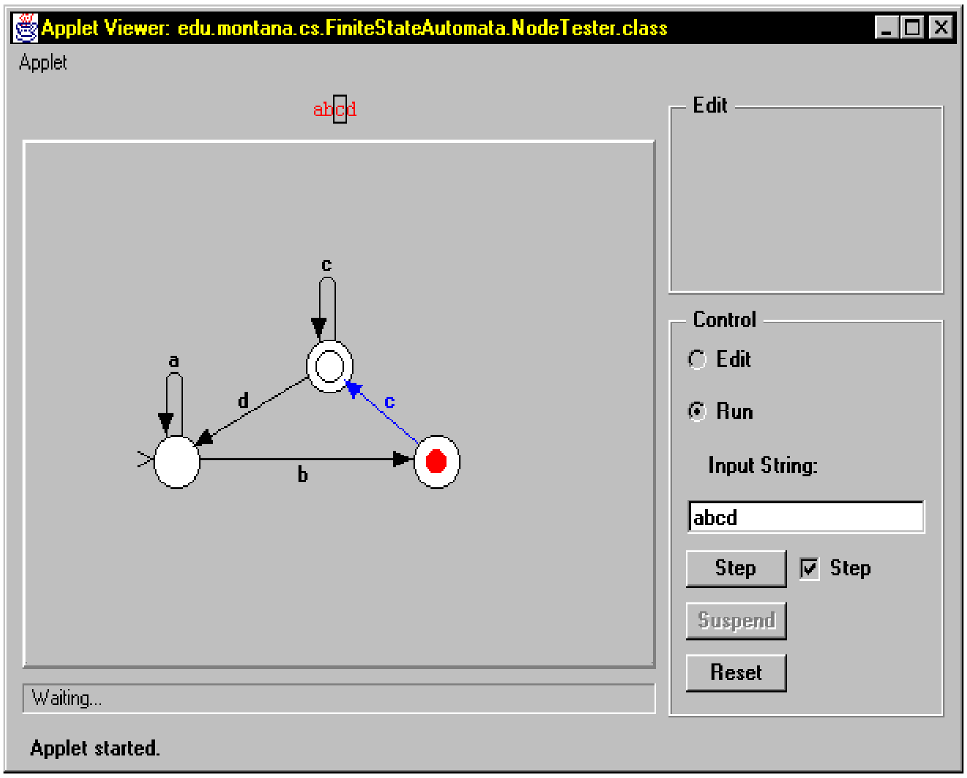

Figure 1: Snapshot of WebLab Finite State Automaton Applet

interactive program animator on the Web, including a Pascal compiler, a graphical user interface for animating Pascal programs, an editor for entering and modifying Pascal programs for animation, a compiler for translating Pascal programs to underlying E-machine code (that can be executed both forward and in reverse), and comprehensive libraries of pre-constructed programs for animation. These are all components of the current DynaLab system that is available on the Web for Microsoft Windows 3.1 (with the Win32s extension), 95, and NT, and described in detail in [2, 4]. With DynaLab, in fact, hypertextbooks are possible, in which the textual material is interlaced with animations of programs or program fragments that can be invoked as desired.

A C/C++ compiler is under development for DynaLab, and consideration is being given to Ada 95 and Java compilers for DynaLab. DynaLab with its extensions to concept and algorithm animations (as discussed in the previous section) will blend to become the WebLab envisioned in this paper. A preview of what the Java version of the DynaLab program animator will look like can be found at

```
http://www.cs.montana.edu/~dynalab
```

Evaluations of Animation Systems

One natural and reasonable question to ask is whether animations are effective as teaching and learning aids. After all, a lot of effort is invested in developing effective animators[3] and animations. Do they work? Intuitively, one would think that they would be extremely effective: if one static picture is worth a thousand words, then surely one could extrapolate to deduce that an interactive animation of a troublesome concept would be worth at least a handful of lectures. However, actual evaluations have produced mixed results in some instances. For algorithm animations, Stasko [7] has reported that although they do help, they may be not much better than a lecture approach, especially after some time has lapsed before a test is administered.

In contrast, our own evaluations of the DynaLab program animator in concert with Stasko's Polka algorithm animator have shown marked positive effects. For example, one evaluation of how well students grasped the concept of open hashing proceeded as follows:

1. Students were asked to read about open hashing in the class textbook.
2. A few lectures on open hashing were then given.
3. In a subsequent two hour formal laboratory session, students were given a pre-quiz on open hashing with a number of keys to hash.

[3] The DynaLab project is currently estimated to consist of over 70,000 lines of code.

4. Students were then required in lab to use DynaLab to animate a Pascal program that implemented open hashing, entering a set of keys and watching how they were hashed.

5. Following the use of DynaLab, students were required to watch an animation of open hashing using Polka,[4] which gave a visualization of the process without displaying any program code; again, they had to hash a set of keys that they were required to enter by hand into the animator.

6. At the end of the lab, students were given a post-quiz identical to the pre-quiz in format (with different keys to hash, of course); they were not allowed to use either animator for this quiz.

The results of the pre-quiz are shown in the following table. There were 11 possible points on the quiz. The first column lists the number of points from 0 to 11. The second column shows how many students earned that many points on the quiz. Notice that 7 students didn't earn a single point. On the other hand, 24 students (out of 62) had learned enough from the book and lectures to earn 10 or 11 points on this quiz.[5] The quiz average was 61.5%.

Pre-Quiz		
Points	**Students**	**Total**
0	7	0
1	1	1
2	6	12
3	3	9
4	0	0
5	8	40
6	1	6
7	2	14
8	2	16
9	8	72
10	14	140
11	10	110
6.77 **(61.5%)**	**62**	**420**

The results of the post-quiz are shown next. The quiz average jumped to 86.5%. Only a handful of students were still clueless, and fully 49 out of 62 earned 10 or 11 points. We like that.

[4]This Polka animation was developed by the last author of this paper

[5]This statistic would seem to bear out our intuition from long observation that the better students are able to visualize or otherwise grasp concepts without much outside help, whereas average and poorer students benefit greatly from additional help.

Post-Quiz		
Points	**Students**	**Total**
0	0	0
1	1	1
2	2	4
3	0	0
4	1	4
5	2	10
6	1	6
7	2	14
8	1	8
9	3	27
10	23	230
11	26	286
9.52 **(86.5%)**	**62**	**590**

A similar evaluation of closed hashing was later performed with nearly identical results.

There are many different evaluations that could be performed, but these and others we have done have convinced us of the effectiveness of program and algorithm animations on learning. Why would our evaluations show more positive benefit than those done by Stasko? It would be hard to say at this point without a clearer understanding of the differences in evaluation techniques. However, it could well be that the combination of the program animator of DynaLab and the algorithm animator of Polka was an important factor. Students not only got the "intuitive picture" by using Polka, but they also got the details of how hashing worked very specifically by watching the program animator of DynaLab. In other words, they got to see both the forest *and* the trees in the laboratory session.

A final comment about the results of these evaluations is this. We believe from our own evaluations that well-designed animations provide a great advantage to learning. However, even if certain animations are shown to be just equal in effectiveness to lectures, we will be tickled. Why? Because we see major applications of Web-based animation systems to distance learning (see the next section) where students will not have the luxury of live lectures and personal contact with an instructor. In these cases, effective animations will play a crucial role.

MSU: Fertile Ground for WebLab

Our situation at MSU illustrates clearly why we are so excited about the prospects of computer science animation, particularly in the context of the World Wide Web. First, there is our own curriculum [10]. An innovative four-semester, integrated introduction to the science of computing is combined with formal laboratories in which students actively experiment with the fundamental concepts of the discipline of computing. A laboratory infrastructure based around program, algorithm, and concept animations makes the laboratory experience[6] meaningful.

Beyond our own doors, though, there are exciting new horizons. MSU has become part of the Western Virtual University, a consortium that should eventually make a university education available outside the confines of traditional ivory towers. We are also beginning to offer remote degrees with two-way interactive video in other locations. Finally, Montana is a rural state with many very small high schools[7] in remote locations. While larger schools often offer computing classes, there is little hope of such opportunities in most of the schools in the state. With access to the Internet and the World Wide Web, such offerings may become reality through connections to MSU. DynaLab has already been downloaded from our Web site and used in high schools in other states, without our having made any efforts at advertising its possible use in pre-college settings (we didn't think it was quite ready for that yet).

Summary

The title of this paper is *WebLab!* It was meant to convey a little of our excitement over the emerging technologies on the Web that promise to make program, algorithm, and concept animations universally available. With these, we have the infrastructure for constructing comprehensive computer science laboratory experiences for students of all ages, reaching those in public schools, universities, and beyond. Online textbooks that integrate hypertext presentations of material—allowing students to progress at their own pace and degree of understanding—with Java applets that animate key programs, algorithms, and concepts are no longer pipe dreams. These in conjunction with the World Wide Web will ensure that traditional walls will come down and that educational opportunities will be made more widely available. These are exciting times. Keep watching!

References

[1] A. Michael Berman. Private Communication, July 1996.

[2] Michael R. Birch, Christopher M. Boroni, Frances W. Goosey, Samuel D. Patton, David K. Poole, Craig M. Pratt, and Rockford J. Ross. DYNALAB: A Dynamic Computer Science Laboratory Infrastructure Featuring Program Animation. In *Twenty-sixth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 27, number 1, pages 29–33, March 1995.

[6] Just as other disciplines, such as physics, have laboratory techniques, tools, and materials that make it possible to provide students with useful experiences in a formal laboratory session, computer science also needs such an infrastructure; we view well-developed animators as the major components of this infrastructure.

[7] The third author of this paper graduated in a class of 9!

[3] Stephen A. Blythe, Michael C. James, and Susan H. Rodger. LLparse and LRparse: Visual and Interactive Tools for Parsing. In *Twenty-sixth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 27, number 1, pages 370–371, March 1995.

[4] Christopher M. Boroni, Torlief J. Eneboe, Frances W. Goosey, Jason A. Ross, and Rockford J. Ross. Dancing with DynaLab: Endearing the Science of Computing to Students. In *Twenty-seventh SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 28, number 1, pages 135–139, March 1996.

[5] Mark H. Brown. Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom. Technical Report 142, DEC Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, May 1996.

[6] David Carlson, Mark Guzdial, Colleen Kehoe, Viren Shah, and John Stasko. WWW Interactive Learning Environments for Computer Science Education. In *Twenty-seventh SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 28, number 1, pages 290–294, March 1996.

[7] Andrea Lawrence, Albert Badre, and John Stasko. Empirically Evaluating the Use of Animations to Teach Algorithms. Technical Report GIT-GVU-94-07, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, March 1994.

[8] Thomas L. Naps. Private Communication, May 1996.

[9] Thomas L. Naps and Jeremy Stenglein. Tools for Visual Exploration of Scope and Parameter Passing in a Programming Language Course. In *Twenty-seventh SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 28, number 1, pages 305–309, March 1996.

[10] John T. Paxton, Rockford. J. Ross, and J. Denbigh Starkey. A Methodology for Teaching an Integrated Computer Science Curriculum. In *Twenty-Fifth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 26, number 1, pages 1–5, March 1994.

[11] Susan H. Rodger. An Interactive Lecture Approach to Teaching Computer Science. In *Twenty-sixth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 27, number 1, pages 278–282, March 1995.

[12] Susan H. Rodger. Private Communication, April 1996.