Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resources for the Web[1]

Rockford J. Ross[1] and Michael T. Grinder[2]

[1] Computer Science Department, Montana State University
Bozeman, MT 59717, USA
ross@cs.montana.edu
[2] Computer Science Department, Montana Tech of the University of Montana
Butte, MT 59701, USA
grinder@cs.montana.edu

Abstract. Computer-generated visualizations have been used in computer science education for many years, most notably in the form of algorithm animations. Although appealing and often useful, the anecdotal evidence is that these visualizations are seldom used in the classroom. There are many reasons for this, including platform dependence, cumbersome installation and maintenance procedures, and—perhaps most influential—a lack of integration with other course materials. Hypertextbooks provide one solution to these problems. Designed as complete teaching and learning resources for the web, hypertextbooks incorporate many features for teaching and learning that vastly extend the capabilities of traditional textbooks. Along with traditional textual presentations of the material to be learned, hypertextbooks allow for different learning paths through the material for different learning needs, an abundance of pictures and illustrations, video clips where helpful, audio, and—most importantly—interactive, active learning visualizations of key concepts. In this paper we discuss the hypertextbook concept by way of the hypertextbook project currently underway at Montana State University.

1 Introduction

Visualizations play a key role in providing insights into important concepts. Computer-based visualizations have been applied to many areas of science and engineering, enhancing our understanding of molecular structures, mysteries of the universe, predator-prey relationships, and many other science, engineering, and sociological phenomena. Most visualization software has been oriented towards advancing research. Less attention has been paid to the development of visualization software for education.

[1] Support for some of the work described here has come from the National Science Foundation, grant numbers NSF-0088728 and NSF-0088934.

At first glance, it might appear that visualization systems developed for research would be equally applicable to education, but that is not the case. Although demonstrations of advanced visualizations can be helpful in the classroom, there is a vast difference between a visualization intended for an expert, who already understands the field well and knows what patterns to look for in a visualization (and what those patterns might mean), and one intended for a novice who does not understand the field well and is using visualization software to help learn the field.

Computer science educators have developed a number of visualizations of computer science concepts for education since the early days of the discipline [1, 13, 17, 15, 8]. Most of these, not unreasonably, are algorithm animations (see, for example, [4, 11, 14])—visualizations that show the steps of an algorithm, such as quicksort, in action. Educators have always struggled to convey the dynamics of an algorithm in a lecture; the artistic and acting capabilities required for an illuminating presentation at a whiteboard elude most instructors. It is also difficult to avoid mistakes, and it is a struggle to back up in such a live presentation to answer student questions about what happened a few steps earlier. Finally, when students walk out of the classroom they leave the dynamic presentation behind. Notes they might have taken, being inherently static, are of little use in recapturing the dynamic information of the lecture.

It is no wonder, then, that the idea of dynamic, computer-based visualizations of key computer science concepts is so appealing to educators. Done properly, they are error free, repeatable, easy to reverse in answer to questions, usable for study outside of the classroom, and readily available to both instructors and students. In spite of their appeal, however, it is well known that visualization software for computer science education is not widely used in instructional settings. In the rest of this paper we discuss the reasons for this paradox and present one remedy: the hypertextbook.

2 Why Educational Visualization Software is Underused

What is it that keeps visualization software systems—even good ones—designed to aid teaching and learning at bay? In [3] we examine this question in depth. Essentially, there are four parts to the answer to this question: (1) platform dependence, (2) installation and maintenance chores, (3) demands on faculty time, and (4) a lack of courseware integration.

The problems associated with platform dependence and installation and maintenance chores are self-evident. These problems can be overcome with well-designed visualization systems designed to run as applets on the web. Such applets are platform independent by definition, and they require no installation or maintenance on the part of the users. However, web-based visualization applets still require time on the part of instructors to locate, evaluate, learn, and teach in preparation for effective classroom use, a process that must be repeated for each desired visualization. Since most such applets deal with single concepts and are therefore useful for only one or two lectures, this is time that most overburdened

faculty members are unable to invest. Finally—and most importantly—there is the problem of courseware integration. An individual applet acquired from the web for visualization purposes is unlikely to blend well with traditional course resources or other applets. Terminology may differ, the implementation of the concept may not closely match that presented in the course textbook, and it may be difficult to decide where to schedule the presentation and use of the applet in the course. Furthermore, individual applets retrieved from the web seldom provide comprehensive active learning experiences for students.

On one hand, then, it is no wonder that visualization software designed for education is underutilized. On the other hand, a solution to this problem is evident. Teaching and learning resources need to be developed that are platform independent and that incorporate active learning visualization applets as a seamless part of the whole. This brings us to the concept of the hypertextbook.

3 Hypertextbooks

A *hypertextbook* is a comprehensive, web-based teaching and learning resource that is intended to augment or supplant a traditional textbook for an academic subject. For example, a hypertextbook on the theory of computing would be a complete, web-based resource for teaching and learning the theory of computing. Hypertextbooks extend the capabilities of traditional textbooks tremendously in that, beyond mere textual presentations and static illustrations, they can also incorporate video clips, audio files, and active links to other material on the web. They can also be arranged (through the use of hyperlinks) to accommodate various teaching/learning needs and styles. Most unique, though, is their capacity for including active learning modules in the form of interactive applets that visualize important concepts and engage students in exploratory learning. In most cases, the visualizations are animated. That is, the concept or model being visualized changes over time in response to various stimuli; we often refer to such visualizations as *animations*. It is this capacity that web-based hypertextbooks have for the incorporation of active learning animation applets that we consider at length in this paper.

The ensuing discussion of hypertextbooks is based on our own work-in-progress on a hypertextbook we call *Snapshots of the Theory of Computing*, or just *Snapshots* for short. The title *Snapshots* reflects the fact that the hypertextbook is being made available in parts (snapshots), as each part becomes ready.

We have reported on *Snapshots* before [2, 3]. Here we present the hypertextbook concept from a different point of view. As the design of *Snapshots* is discussed, we will highlight points that we have found to be important in the construction of a hypertextbook, both from pedagogical and practical points of view. For ease of reference, we will number these points.

It is important to note at the outset that none of the "important points" we list have actually been verified by us through formal studies with hypertextbooks in a teaching and learning environment; we have not yet had opportunity to

conduct such studies[2]. Instead, the points listed come from our own experience, discussions with colleagues, presentations by cognitive psychologists, and the literature (for example, [14]).

It is equally important to acknowledge that many of the points we highlight—even though we arrived at most of them independently—are not unique to our work. The use of hypertext and hypermedia in teaching and learning has been investigated over the course of a number years (as an example, see [10]). The idea of constructing hypertext teaching and learning resources with integrated visualizations and animations is also, of course, not unique to our project. In [5], for example, eleven "design principles for effective web-based software visualizations which cover teaching requirements, sustainability, ease of use, and remoteness" are discussed based on the work of the authors, who build on earlier work [6]. On the other hand, we know of no other work currently in progress that captures the essence of our hypertextbook project, which focuses on the inclusion of comprehensive, integrated, and animated active learning applets.

With these acknowledgements to the substantial work of others, we (for the sake of brevity) will make few more references to the literature in presenting the list of issues we have found to be important in the design of a hypertextbook.

3.1 The Hypertextbook Cover

Figure 1 provides a view of the "cover" of *Snapshots* as it appears when viewed in the Netscape web browser[3]. This is the home page of the web that makes up the *Snapshots* hypertextbook and thus serves as the portal to the book. One will notice that this page does indeed appear similar to the cover of a traditional textbook. We have attempted to make it attractive, uncluttered, and functional, which leads us to our first points.

> **Point 1.** The cover, or portal, of a hypertextbook should be attractive and/or intriguing, thus inviting readers to explore further.

> **Point 2.** A hypertextbook, from the cover on, should have a familiar and professional "textbook" look to it, so that students feel comfortable using it as their main class learning resource. (As time goes on and hypertext materials become more prevalent, it is likely that such visual relationships to traditional textbooks will become unnecessary.)

> **Point 3.** A hypertextbook should be uncluttered, yet functional. There is a strong temptation to make liberal use of the "bells and whistles" available for web page development in a hypertextbook (flashing symbols, odd fonts, animated images, and so forth), most of which only confuse the learner and detract from the learning experience.

[2] This is the classical "chicken and egg" problem; such studies would help in the design of a hypertextbook, but one needs a hypertextbook to conduct the studies.

[3] It is, unfortunately, not possible to reproduce colors here. Where important we will explain the color schemes in the figures.

We tried a number of different designs before recognizing the importance of points 1 through 3.

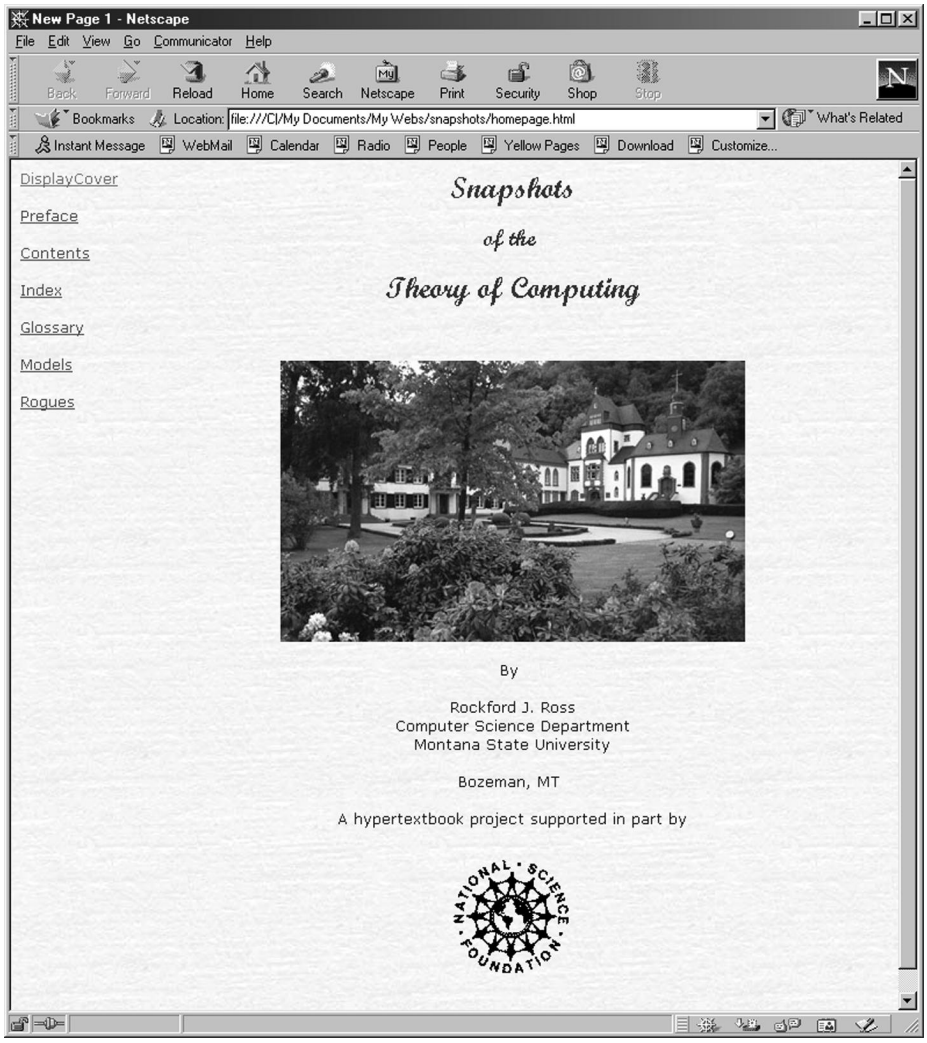

Fig. 1. The cover of the *Snapshots* hypertextbook

As seen in figure 1, the main entry points to the hypertextbook are listed as links along the left side of the cover. This left margin with its links appears on every page of the hypertextbook, giving students easy access to the important parts of the book: cover page, preface, contents, index, glossary, standalone versions of the active learning visualization applets (the "Models" link) used in

the book, and a list of the book's contributors (the "Rogues" link). Fonts, page backgrounds, and color schemes are also used consistently throughout the book.

> **Point 4.** A hypertextbook should maintain a consistent structure throughout. Standardized page design, common font usage, and consistent application of color schemes are all important.

3.2 The Hypertextbook Structure

The web provides opportunities for structuring a hypertextbook in ways that traditional textbook authors can only dream about. As examples, a hypertextbook can be organized—through the use of hyperlinks—to have different learning paths (through the same material) for different learning styles or for different levels of educational maturity. In *Snapshots* we have chosen the latter approach, creating paths through the book that cater to novices, intermediate learners, and advanced students, respectively. From the "Preface" link in the left margin one can get to a description of the organization of *Snapshots* which displays the illustration given in figure 2.

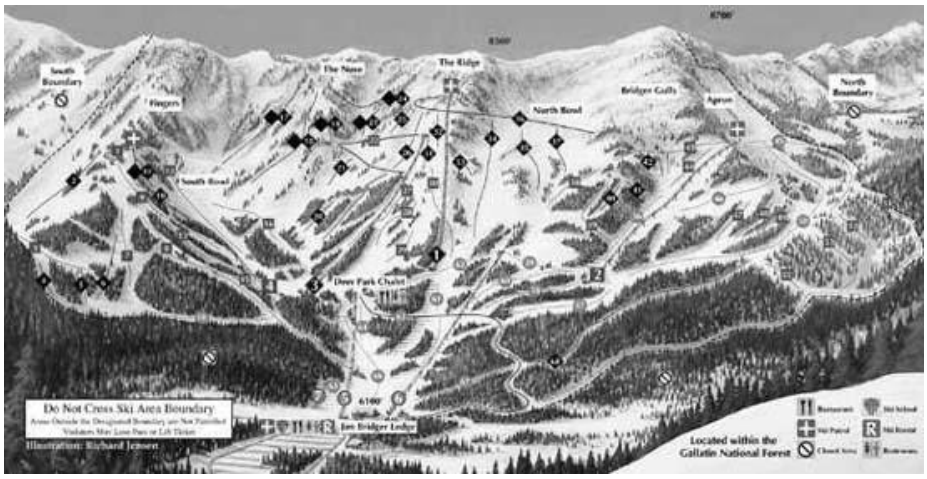

Fig. 2. The ski slope organizational model of *Snapshots*

Figure 2 is a copy of a ski trail map for Bridger Bowl, a ski destination in the Rocky Mountains near Bozeman, Montana, USA. One can see that there are many trails that start at various high points on the mountain. All trails are marked with international symbols to make it possible for skiers to choose the appropriate way down based on their abilities. Green circles mark the easiest routes, blue squares the intermediate paths, and black diamonds the challenging trails. All lead to the same goal: the lodge at the bottom.

The ski slope model is our inspiration for the organization of *Snapshots*. We are designing the book, as noted, to address the needs of beginners, intermediate learners, and advanced students. We mark the different ways through the book with the same international symbols used on ski slopes: a green circle for beginners, a blue square for intermediate learners, and a black diamond for the more advanced. All paths lead to the same goal: an understanding and appreciation of the theory of computing. This structure makes the book usable across the curriculum.

Point 5. A hypertextbook should make use of hyperlinks to organize the material in ways appropriate for different learning needs and/or different learning styles to make it as flexible and useful as possible.

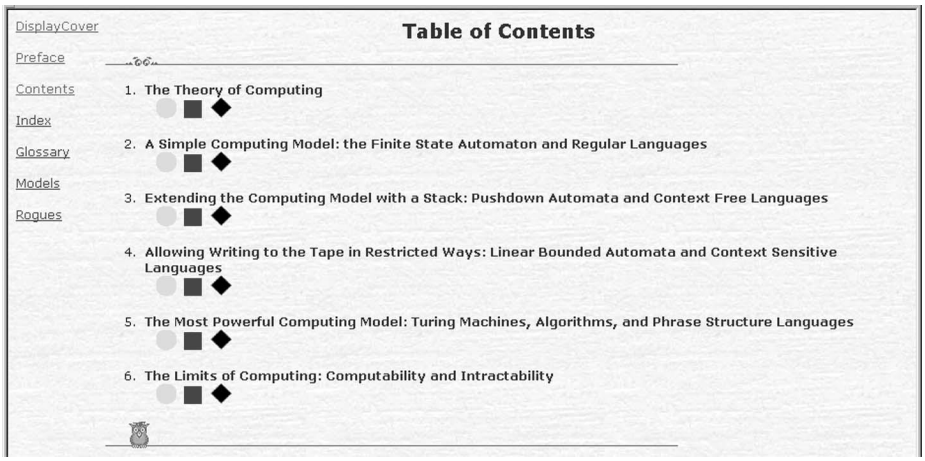

Fig. 3. The contents page of *Snapshots*

To see how this works, look at figure 3, which is a clip of the page reached by selecting the "Contents" link in the left margin. Notice that beneath each chapter title the three international symbols appear: the (green) circle, the (blue) square, and the (black) diamond. These symbols have associated hyperlinks that lead the user to a more detailed table of contents for that chapter based on the level selected. From the more detailed table of contents a student can then begin to learn about the topic of the chapter by clicking on the desired section of the chapter (each of which is also a hyperlink). The pages for each topic are marked at the top by the appropriate international symbol to let the students know whether they are on the appropriate track.

Following the green circle route leads a student through a very intuitive presentation of the topic being studied, with lots of examples and liberal use of the animated, active-learning applets that we have designed for the theory of

computing. The blue square track also provides an intuitive introduction to the topic, but with fewer examples involving the active-learning applets and a greater reliance on mathematical notation. The black diamond approach incorporates only a few examples that use the active learning applets and resorts to formal mathematics throughout.

3.3 Animated, Active Learning Applets for Hypertextbooks

At this point we can finally discuss the central feature of the *Snapshots* hypertextbook—animated, active learning applets of the key concepts of the theory of computing. It might seem that these could be discussed in isolation, but doing so would obscure one of the most important issues of hypertextbook design: applets designed for use in a hypertextbook have substantially different requirements than those designed for standalone use. They also take much more time to create. An old subjective metric [7] states that if a software system designed for personal use takes time N to complete, the same system designed for use by others will take time $3N$, and if it is also to be integrated with another system (e.g., a hypertextbook) it will take time $9N$. Our experience certainly lends credence to this observation.

> **Point 6.** Creating animated, active learning applets that integrate well with each other and fit seamlessly into a hypertextbook takes appreciably more time than creating standalone applets for identical topics.

By way of introduction, look at figure 4. This is a page that appears towards the end of a section in *Snapshots* that discusses nondeterministic finite state automata on the green track. Notice that the applet is embedded directly in the text of this page. We have discovered that opening a new window for an applet is distracting to the student, in that it causes focus to be shifted away from the discussion in the text. It is also confusing in that students are not sure when a newly opened window should be closed, nor is it clear when or how to return to the text. On the other hand, standalone versions of the applet should also be available for students to use in their own explorations (this feature is provided in *Snapshots* through the "Models" link in the left margin).

> **Point 7.** Applets that are to be used in examples in a hypertextbook should appear in line and not be displayed in new windows.

> **Point 8.** Applets used in a hypertextbook should also be available (in an appendix, for instance) in standalone mode for arbitrary student or instructor use.

Figure 4 shows the automaton applet preloaded with an example nondeterministic automaton for determining whether an input string is a valid integer, fixed point, or floating point number. Configuring the applet for this particular use is, of course, the responsibility of the hypertextbook author.

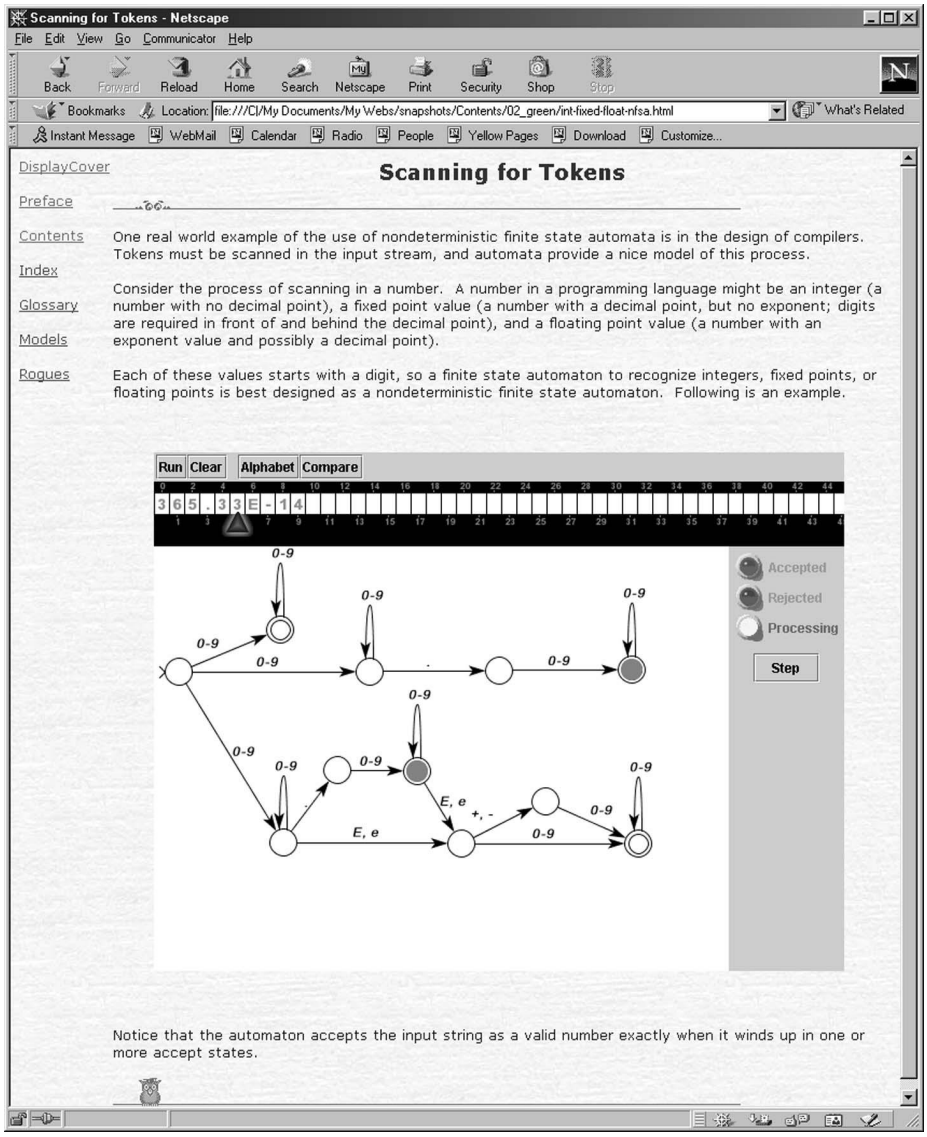

Fig. 4. An embedded finite state automaton applet in *Snapshots*

Point 9. A special software tool must be provided with each applet that allows an author to configure that applet properly for each appearance of that applet in the hypertextbook. For instance, if the applet in question is to illustrate a particular finite state automaton in an example at some desired point in the hypertextbook, the author must be able to configure the applet to start up with the desired automaton at that point.

Active learning applets should have a number of special features that aid learning. One is a feature that can lead a student through an animation a step at a time with little or no required intervention and with accompanying explanations. This feature is important when examples of a new topic are encountered for the first time. In the case of the finite state automaton applet, for instance, this would mean that the applet would be preloaded with both an automaton and an input string. Each time the student presses the "Step" button, the automaton would be shown consuming the current input symbol, changing states, and moving the input head to the next input symbol. An explanation of this step would appear simultaneously in another pane of the applet window.

A second feature of applets intended for active learning is one that gives students more responsibility for directing the animation. Again using the finite state automaton applet as an example, in this instance an automaton might be preloaded into the applet by the author, but the student would then provide his or her own input strings and run the automaton to see whether the strings were accepted or rejected.

A final necessary feature for active learning applets is one that assigns complete responsibility to the student (for example, when an applet is used in an exercise). In the case of the finite state automaton applet, the student should be able to create and modify an automaton arbitrarily within the applet in solving a given exercise or for independent exploration.

Other important features include the capability to back up in an animation (so that the student can explore puzzling aspects of the animation), and an option to set the animation either to proceed one step at a time under student control or to set it to run automatically with an accompanying method (e.g., a slider bar) for controlling the speed of automatic execution of the animation.

Point 10. Active learning animation applets intended for use in a hypertextbook must provide a wide range of control to the student, from virtually no control (so that a new concept can be explained a step at a time by the author), to intermediate control (so that a predetermined example can be explored by a student controlling the animation of that example), to complete control (so that a student can construct and control an animation of a concept from scratch in an exercise). It has been shown that the more involved a student is in creating an animation, the better the student learns [16, 14].

Point 11. Active learning applets should provide capabilities for a student to control each step in an animation or to set the animation to run

continuously; in the latter case, there should be a mechanism that allows the student to adjust the speed of automatic animation.

Point 12. Active learning applets should allow a student to back up arbitrarily far in an animation to review or retry certain steps.

In figure 4 the nondeterministic automaton is shown part way through the processing of the input string `365.33E-14`. There are three nondeterministic branches in this automaton that check whether the input string is an integer, fixed point, or floating point number, respectively. The current states are shaded (with a red disk). As an input symbol is consumed, a state transition is shown in that the red disks move smoothly from their current states simultaneously across the appropriate transition edges to the next states (if there is no corresponding transition from a state, the red disk turns gray and then disappears). The input head also moves to the next input symbol. State transitions can be controlled a step at a time through the "Step" button or set to run automatically. Rudimentary sound effects accompany these actions; there are different sounds for state transitions, string acceptance, and string rejection that draw attention to these activities and distinguish them from one another. One can see that the graphical version of the automaton reflects closely the form of automaton models found in traditional textbooks.

Point 13. Animated, active learning applets should provide smooth transitions between images, or states, in the animation. Students can then see more easily how the step being animated occurs.

Point 14. Sound should be used where appropriate in active learning applets. Sound can give important clues to an animation, which today's students are accustomed to utilizing, for example, while playing computer games, and even while interacting with general software systems (e.g., operating systems).

Point 15. Models animated in active learning applets should not deviate in appearance substantially from their traditional visual representations unless there are sound pedagogical reasons for a change.

3.4 Integrating Applets in Hypertextbooks

The repertoire of applets designed so far for *Snapshots* includes those needed for a complete first chapter of a traditional theory book on finite state automata, regular grammars, and regular expressions. In addition to the finite state automaton applet shown in figure 4, there is a context free grammar applet that allows an author to illustrate arbitrary context free grammars (and hence arbitrary regular grammars) and animate derivations with these grammars. Students can use this applet in exercises to create grammars and to construct derivations of arbitrary strings in those grammars.

There is also a regular expression grammar that an author can use to demonstrate the construction of a regular expression for a provided regular set. It also allows a student to construct regular expressions.

Finally, there is a program animator that animates (Pascal) programs. It can be used to demonstrate implementations of various algorithms related to the theory, such as how a finite state automaton can be implemented as a program, how a regular expression is converted to a finite state automaton, and so forth. (in this instance, Pascal serves as a nice pseudo-language). These applets are not illustrated here for lack of space, but can be found at [12].

The finite state automaton, context free grammar, and the regular expression applets have been designed to work together to provide feedback to a student completing exercises using these applets[4]. Using known properties of finite state automata, the automaton applet incorporates an algorithm that checks a student-constructed automaton for accuracy. To accomplish this, the hypertextbook author providing the exercise also gives a correct finite state automaton (hidden from the student) for the exercise. The language of the correct automaton is then compared with the language of the student's automaton. If the two are equal, the student is congratulated. If the two languages are not equal, the applet reports this to the student along with a sample string that the student's automaton either accepts or rejects in error.

Using other known algorithms that convert regular expressions and regular grammars to equivalent finite state automata the same technique is applied in the regular expression and grammar animation applets to provide similar feedback to a student. Again, the author must provide a correct regular expression or grammar, respectively, when creating exercises with these applets. Then the conversions to equivalent finite state automata of the correct version and the student version are made, the language comparisons completed as above, and feedback provided to the student.

> **Point 16.** Active learning models included in a hypertextbook must be designed to interact with each other as appropriate.

3.5 Decoupling a Model from its Description

This brings us to our final point, and perhaps the most important one we make.

> **Point 17.** The data structure that describes a model (e.g., a finite state automaton) must be independent of its graphical representation in an applet.

This last point is easily overlooked. It is the reason why many standalone applets cannot readily be extended, or incorporated into other resources, such as hypertextbooks. Without an underlying representation for each model that is independent of its graphical presentation, integrating applets to work together (as

[4] Much of this work is in progress at the time of this writing.

described in the previous section) would be prohibitively difficult. With *Snapshots* we have implemented the most logical solution to this problem by designing for each model (finite state automaton, grammar, and regular expression) an eXtensible Markup Language (XML) definition. Not only does this allow the various models to be integrated with each other, but it also provides for the development of different graphical representations of the models as desired, and it allows the models to be treated in a fashion that is now standard on the web.

Briefly, this works as follows. Consider the definition of a finite state automaton. This definition can be formally specified in XML so that all of the components of a finite state automaton—the states, the input alphabet, the transition function, and the accept states—are well defined. Each different finite state automaton is formulated with the same XML structure, but with different values in the data fields (e.g., the fields that represent the number of states, the actual input alphabet, and so forth). In our case, where pedagogy is key, there may also be other parts to the definition not normally associated with the theoretical definition of a finite state automaton, such as a field that accompanies each state that describes what that state "remembers," or a particular input string on which the automaton should be started when it is initialized. Along with the XML file for a finite state automaton, a Document Type Definition (DTD) file is provided that specifies the rules for constructing a proper finite state automaton in XML. Thus, programs that process the XML file can check it against its DTD to see that the file represents a correctly structured finite state automaton before actually processing the XML file.

The key point is that the XML file for a finite state automaton contains only its definition, not any information about how the automaton is to be displayed. Thus, it is up to the program processing an automaton XML file to decide how to display the automaton. The illustration in figure 4 represents one way to display a finite state automaton from its XML file. The same automaton could be displayed as a table from the same XML file, since no information is included in the file about how to display or animate the automaton. On the other hand, since all of the information about the automaton is in the XML file, it is possible to display the automaton consistently in one of these various forms, and even to animate it as desired.

Consider also what is done when an author creates an exercise for the hypertextbook using an exercise creation tool (see point 9). Suppose this exercise requires a student to construct a finite state automaton to recognize a particular regular language. In developing the exercise, the author provides both the written specifications of the automaton for the student and a correct automaton. This correct automaton is automatically converted to its XML form by the exercise construction tool and stored as part of the exercise (hidden from the student). The student completing the exercise then attempts to construct a correct automaton within the finite state automaton applet. Behind the scenes, the applet converts the student's automaton to its respective XML form. When the student then clicks on a "submit" button, the applet invokes an algorithm that compares the correct automaton against the student's automaton using a well-

known algorithm for determining whether two finite state automata recognize the same language. This algorithm is easy to implement because of the consistent representation of both automata as correct and consistent XML files (each based on the same DTD).

The context free grammar and regular expression models in *Snapshots* also have appropriate, consistent XML representations. This makes implementation of standard conversion and checking algorithms (e.g., algorithms for checking whether a student-constructed regular expression is correct or not, or for converting a regular expression to a finite state automaton) in the active learning applets that animate these concepts straightforward as well.

Based on the XML definitions for the different models, work is continuing as of this writing on extending the active learning applet library of *Snapshots* to include animations of the standard conversion algorithms from nondeterministic finite state automata to equivalent deterministic versions, from deterministic finite state automata to their minimal form, from regular expressions to finite state automata (and vice versa), and from regular grammars to finite state automata (and vice versa). Finally, animations of the applications of the theory—such as the pumping lemma for regular languages, the Myhill-Nerode theorem, and others—are in the plans. Together, these animations will support a comprehensive first chapter in *Snapshots* on finite state automata, regular grammars, and regular expressions, replete with animated, active learning applets designed to help students of many different abilities and backgrounds come to an appreciation and understanding of the theory of computing.

4 Summary

In this paper we have discussed some of the problems that have precluded the widespread use of standalone educational visualization software systems in the computer science curriculum. We have then proposed one solution to these problems: hypertextbooks that are designed for the web and are thus platform independent, turnkey systems that can be used as the primary teaching and learning resource in a course. Such hypertextbooks can (and should) incorporate animated, active learning applets in a seamless fashion, so that students use them as a matter of course. Finally, we have listed a number of important points that we have learned in the course of our efforts to construct hypertextbooks in the Webworks Laboratory [12] at Montana State University.

There are a number of researchers working on visualization and animation software for education, including animations of theory concepts (see, for example, [9]). However, we know of no other hypertextbook projects of the scope described here (one project worth a look is [18]). We hope that our work will encourage others to begin similar projects. Patience is required. The writing of a textbook is a big job in any case. Designing a hypertextbook that addresses the differing needs of students and that incorporates active learning applets makes the job much more challenging. In the end we believe that students will profit.

References

1. 16mm color sound film. *Sorting Out Sorting.* 30 minutes, 1981.
2. Boroni, C. M., Goosey, F. W., Grinder, M. T., and Ross, R. J. A Paradigm Shift! The Internet, the Web, Browsers, Java, and the Future of Computer Science Education. In *Twenty-ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)* (Mar. 1998), vol. 30, number 1, pp. 145–149.
3. Boroni, C. M., Goosey, F. W., Grinder, M. T., and Ross, R. J. Engaging Students with Active Learning Resources: Hypertextbooks for the Web. In *Thirty Second SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)* (Mar. 2001), vol. 33, number 1, pp. 65–69.
4. Brown, M. H. Zeus: A System for Algorithm Animation. In *Proceedings of the 1991 Workshop on Visual Languages* (Oct. 1991).
5. Domingue, J., and Muholland, P. An Effective Web Based Software Visualization Learning Environment. In *Journal of Visual Languages and Computing* (Oct. 1998), vol. 9, number 5, pp. 485–508.
6. Eisenstadt, M., and Brayshaw, M. *Understanding the Novice Programmer.* Erlbaum, Hilldsdale, NJ, E. Soloway and J. Spohrer, Eds., 1987, ch. An integrated textbook, video and software environment for novice and expert Prolog programmers.
7. Frederick P. Brooks, J. *The Mythical Man-Month.* Addison Wesley, 1975.
8. Greening, T., Ed. *Computer Science Education in the 21st Century.* Springer Verlag, 2000, ch. Shifting Paradigms: Teaching and Learning in an Animated, Web-Connected World, pp. 173–193. Invited chapter by Rockford J. Ross.
9. Hung, T., and Rodger, S. H. Increasing Visualization and Interaction in the Automata Theory Course. In *Thirty-first SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)* (Mar. 2000), vol. 32, number 1, pp. 6–10.
10. Jacobson, M. J., Maouri, C., Mishra, P., and Kolar, C. Learning with hypertext learning environments: Theory, design, and practice. In *Journal of Educational Multimedia and Hypermedia* (1996), vol. 5, number 3/4, pp. 239–281.
11. Naps, T. L., Eagan, J. R., and Norton, L. L. JHAVÉ — An Environment to Actively Engage Students in Web-based Algorithm Visualizations. In *Thirty-first SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)* (Mar. 2000), vol. 32, number 1, pp. 109–113.
12. Ross, R. J. *Webworks Laboratory Web Site.* http://www.cs.montana.edu/webworks.
13. Ross, R. J. Teaching Programming to the Deaf. *ACM SIGCAPH Bulletin 30* (Autumn 1982).
14. Stasko, J. Evaluating Animations as Student Aids in Learning Computer Algorithms. *Computers & Education 33*, 4 (1999), 253–278.
15. Stasko, J., Domingue, J., Brown, M. H., and Price, B. A., Eds. *Software Visualization: Programming as a Multimedia Experience.* MIT Press, 1997.
16. Stasko, J. T. Using Student-Built Algorithm Animations as Learning Aids. In *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)* (Mar. 1997), vol. 29, number 1, pp. 25–29.
17. Thomas, D. A., Ed. *Scientific Visualization in Mathematics and Science Teaching.* Association for the Advancement of Computing in Education (AACE), 1995, ch. Visualizing Computer Science. Invited chapter by Rockford J. Ross.
18. Wilhelm, R. Ganimal project: Ganifa—electronic textbook on generating finite automata. http://www.cs.uni-sb.de/GANIMAL/, 2001.