Addressing Pedagogical Requirements in Algorithm Visualizations

Guido Rößling
Dept. Electr. Eng. & Computer Science
University of Siegen
Hölderlinstr. 3, D-57068 Siegen, Germany
roessling@acm.org

Thomas L. Naps
Department of Computer Science
University of Wisconsin Oshkosh
Oshkosh, WI 54901
naps@uwosh.edu

Abstract

Although algorithm visualizations have become numerous, they still have not been successfully adapted into mainstream computer science education. Algorithm visualization systems need to better address pedagogical requirements for effective educational use. We discuss the relevance of several such requirements that are not supported in most systems. The combination of two existing algorithm visualization systems implements these requirements and thereby provides a rich testbed for future studies of effectiveness.

1 Introduction

Algorithm visualization (AV) uses computer graphics to depict the actions of an algorithm. Many AV tools have been built and are freely available over the World Wide Web. The reasons for their popularity are threefold. For researchers in the field of AV, there is the challenge of developing new visualization techniques. For practitioners, AV can help in the process of designing and debugging algorithms. For computer science students, AV holds promise to help them understand algorithms more easily and in greater depth. However, despite the abundance of algorithm visualization tools now available, their promise as a pedagogical tool is largely unfulfilled. According to Baecker [2], "little of the work (in algorithm visualization) has been adapted by the mainstream of computer science education and practice." Stasko and Lawrence [16] maintain that "algorithm animations used as passive videos of an algorithm's operations will have minimal impact on learning."

Why is this and what steps can we take to correct it? Many of the current AV systems concentrate on graphics rather than on pedagogy [17]. These systems lack features to encourage students' interaction with the AV system. "Too often in the past, descriptions of visualization systems rarely specified any particular (pedagogical) task they were intended to support" [11].

This paper describes how the merging of two systems, ANIMAL and JHAVÉ, addresses these issues. ANIMAL [13] is an algorithm animation engine that handles the graphics display of the animation. JHAVÉ [9] is an environment for pedagogically staging algorithm animations. Now, by using JHAVÉ to host animations depicted by ANIMAL, we create a synergy that directly meets a unique set of pedagogically-driven requirements heretofore never collectively present in a single AV system.

2 Pedagogical Requirements

Research has shown that, to be an effective learning tool, an AV system must satisfy an increasingly large set of requirements.

Reliably reaching a large target audience. The system's platform should be chosen to allow the widest possible target audience. Currently, this means that the system should be provided as a Java applet or application. The security restrictions placed on applets preclude the access to a local file system or connecting to a different web server during the execution. Applications do not share these restrictions. They therefore allow the storage of information to the local file system, as well as the interaction with multiple animations in a single session. Finally, applets are at the mercy of the Java virtual machine as implemented in browsers such as Netscape and Internet Explorer. As such, they typically cannot take advantage of new features built into the language. Instead, when writing applets, one essentially is forced to write for a "lowest common denominator" of the language. For these reasons, applications are preferable to applets.

General-purpose systems. Another key issue in designing an AV system involves the choice between implementing a general-purpose versus a topic-specific system. Topic-specific systems may offer highly specialized and optimized op-

erations for the target area. However, they are rarely usable outside their limited focus. Users thus may have to adapt to a different system whenever the current topic changes, as it is bound to do within even a single course. General-purpose systems, on the other hand, offer a common interface to a multitude of animations. As general-purpose systems can depict a wide variety of algorithms, instructors are better able to integrate AV into their entire course. Research [3] has shown that such integration is vital to students' becoming comfortable with AV as a learning tool.

Allowing users to provide input to the algorithm. In some cases, students should be able to provide input to the algorithm being visualized. This becomes particularly critical in those situations when we want the student to identify best- and worst-case behavior of algorithms. However, we must be careful that facilities to input data do not overwhelm the student. For example, designing strategic input data for a graph algorithm can be a very time-consuming task. Consequently, data input facilities must be carefully constructed to allow the student to focus on those issues pertinent to the type of understanding we are trying to achieve.

Rewind capability. An AV system that strives to be truly effective for algorithm understanding must present its users with an interface that makes it possible to rewind the algorithm's execution [1]. When a student becomes lost or confused in watching a visualization, she must be able to backtrack to the point where she became lost. Moreover, she must be able to backtrack as far as she needs on a single step basis. Restricting this navigation to restarting the animation from the beginning is insufficient.

Just as the ability to backtrack and re-read is absolutely essential in comprehension of written technical material, it is equally apparent that such abilities are essential to effectively use AV in achieving any non-trivial level of understanding. In a study by Stasko and Lawrence [16], the most often cited negative comment on the part of the participants was the inability to rewind the animation. The participants said that they "often wanted to look at the heap as it appeared before the operation." Despite the importance of a flexible rewind facility, to our knowledge this feature has been neglected in all general-purpose AV systems that present their visualization via smoothly animated motion instead of as a sequence of discrete snapshots.

Structural view of algorithm. Gloor [5] stresses the importance of a structured view of the algorithm's main parts. The student should be able to jump directly to selected key points of the animation by clicking on an element in this view. The feature requires the capability to jump to a specific animation state in either direction.

Interactive prediction. The system must support interactive prediction, that is, interrupting the actual visualization with stop-and-think questions that pop up at an interesting event during the algorithm's execution. Such questions make the student predict what she will see in the next step of the visualized algorithm. Without such questions, once a student becomes confused, continuing to watch a visualization is akin to watching a movie in which one has lost interest [10]. The insertion of questions can change this dramatically. When a confused student answers a question incorrectly, continuing with the visualization not only provides the student with the correct answer but also serves to reset her perception of the algorithm back on the track intended by the instructor.

Integration with database for course management reasons. A recent study by Jarc et al. [8] brought into doubt the value of interactive prediction. Students using interactive prediction did no better than students who were not using the system at all. Jarc hypothesizes that this ineffectiveness is because poorer students merely treated the interactive questions as a guessing game. To correct this shortcoming, we require that our system not only support stop-and-think questions, but also allow the student to enter a "quiz-for-real" mode. In this mode, the student's responses to questions are recorded in a database on the server. Apart from providing valuable feedback for instructors, these results can be used in evaluating students. Hence they naturally heighten the student's level of intensity and attentiveness when using the system. In a study [6] more recent than Jarc's, one group of students "forced" to take such for-real quizzes did better in learning Quicksort than those whose answers to stop-and-think questions were merely used for immediate feedback. As educators, we should not be surprised that making quizzes count in this fashion improves the effectiveness of AV. Bonwell and Eison [4] note that the recall of information eight weeks after the lecture can be doubled by a quiz following the lecture. This may also apply to AV.

Hypertext explanations of the visual display. The system should support the integration of explanatory hypertext to help the student understand the mapping of the algorithm's abstractions to the AV system's display of those abstractions. In [16] Stasko and Lawrence note that except for very straightforward mappings, students often cannot translate the AV system's graphics to the algorithm that is being depicted. They state: "Time after time we witnessed students view algorithm animations in a puzzled manner, unable to decipher the visual mapping. One way to address this problem is to make sure that the visualization itself is thoroughly explained." Uses of accompanying hypertext alluded to by Stasko and Lawrence include viewing the underlying source code, pseudocode, or a explanation analogous to what one would find in a textbook. This documentation may be static or ideally adaptive to the current state of the algorithm and dynamically including actual values used in the current execution of the algorithm [1].

Smooth motion. Several research reports including [1, 15] hint that smooth motions help users detect the change between successive steps. However, some students may prefer the display of discrete snapshots over smooth motions [1].

Additionally, smooth motions become impractical if the data to be manipulated is too large. The AV system should support smooth motions, but also offer the option of viewing the animation in discrete steps. Animation rewinding should also allow the user to choose either discrete steps or smooth displays. In the latter case, the operations have to be performed smoothly in reverse direction.

Reports also indicate that animations without a break between steps may cause problems for students. For example, [1] quotes a student stating that "the animation doesn't wait for you to think". On the other hand, forcing the user to perform a certain action to advance to the next step may become tedious, especially for longer animations. Therefore, the AV system should at least allow the person generating the animation to specify a break between consecutive steps. This break may require a fixed user action or simply wait for a specified time before the animation continues. The progress of the animation must also be pausable.

3 Related Work

Many AV systems are freely available. We therefore select a few representative systems to outline the typical achievements and shortcomings regarding our requirements. All systems are implemented as Java applets or applications and can therefore reach a large target audience.

JAWAA [12] is a scripting-based AV system implemented in Java. JAWAA offers general graphic primitives and effects. Several specialized commands indicate that is not geared as a general-purpose system. These commands address typical data structures including arrays, trees, stacks and queues. The user can view the animation as a slide-show without breaks between steps or on a single step basis. The animation author can choose between discrete and smooth animation effects. However, only the user can adjust the speed of the smooth animation effects. The author of an animation may add arbitrary text at any time and place. As components use absolute coordinates, positioning components next to other objects requires coordinate calculations. JAWAA does not support explanations styled as HTML-hypertext. It also does not support rewinding, reverse playing, a structural view of the animation or interactive predictions.

JSamba [14] shares the main characteristics of JAWAA. It is also scripting-based, but does not offer specialized commands for data structures. The author may spread the animation over several different views. JSamba is a Java-based simplified front-end for the animation engine employed in the widely-known XTango [15] system, which offers an API for generating animations.

IDSV [8] is an applet-based system that is not general-purpose. Rather it has a built-in set of algorithms that depict sorting, binary trees, and a small number of graph problems. It offers both smooth animation and a discrete-step mode. When the student watches the animation in IDSV's "I'll try" mode, she is prompted with questions asking her to predict will happen next. Some recording of the student's progress in answering these questions is done by the system to allow instructor analysis of students' progress.

ANIMAL [13] offers three distinct ways of generating animation: by scripting, using a special Java-based API, or within a GUI. Explanatory text can be easily added at any point to an animation. Relative object placement allows for easy positioning without calculating positions. ANIMAL lets the user step to any point within the animation in either direction. It originally did not support a structured view of the animation, accessing external hypertext documentation, or performing interactive predictions.

GAIGS [9] is a scripting-based visualization system that portrays an algorithm as a discrete sequence of data structure snapshots. Unlike many other systems, the scripting syntax of GAIGS specifies data structures (arrays, matrices, linked lists, stacks, queues, binary trees, general trees, and graphs) instead of graphical primitives. GAIGS takes care of rendering such structures on the screen. This often makes it easier to quickly develop a visualization in GAIGS since one may think in terms of data structures instead of graphics. GAIGS allows its user to move forward and backward through the sequence of snapshots. However, GAIGS does not support smooth motion, and, because it does its own layout of data structures, it can limit the graphic creativity of the visualization developer.

JHAVÉ [9] is not itself an algorithm visualization system but rather a platform offering built-in pedagogical aids to visualization systems that meet certain requirements. Essentially these requirements are that the visualization system (1) is written in Java, (2) uses textual scripting to describe the animation, and (3) can be adapted to JHAVÉ's *Visualizer* interface. This interface consists of a *readScript* method responsible for parsing an animation script and a *runScript* method responsible for rendering the animation. In return for this, JHAVÉ provides such *Visualizer* objects with a client-server architecture in which the *Visualizer* becomes the client, connecting to a JHAVÉ server that:

- delivers hypertext documentation in the form of an HTML window linked to URLs on the server,
- provides input generators that allow users to provide input to the algorithm being portrayed by the *Visualizer*,
- augments the scripting language of the *Visualizer* to allow for the insertion of stop-and-think questions during the animation,
- integrates the *Visualizer* with server-side course management facilities that record the grades and responses of registered students using the stop-and-think questions as a "for-real" quiz.

As described in [9], JHAVÉ *Visualizer*s had previously been

developed for the GAIGS and JSamba scripting languages. Inherent limitations of both of these scripting languages made JHAVÉ a pedagogic platform looking for a better scripting language to take fuller advantage of the tools it offered. ANIMAL provides that. With relative ease, ANIMAL was adapted to JHAVÉ 's *Visualizer* interface. The result is a system whose advantages are compared to the other systems in Table 1.

System	general	input	rewind	structure	prediction	database	hypertext	smooth
JAWAA	(✔)	-	-	-	-	-	(✔)	(✔)
JSamba	✔	-	-	-	-	-	(✔)	(✔)
IDSV	-	✔	-	-	✔	(✔)	-	✔
GAIGS	(✔)	-	✔	-	-	-	-	-
ANIMAL	✔	-	✔	-	-	-	✔	✔
JHAVÉ+ ANIMAL	✔	✔	✔	✔	✔	✔	✔	✔

Table 1: Requirements addressed in AV systems. (✔) indicates partial support

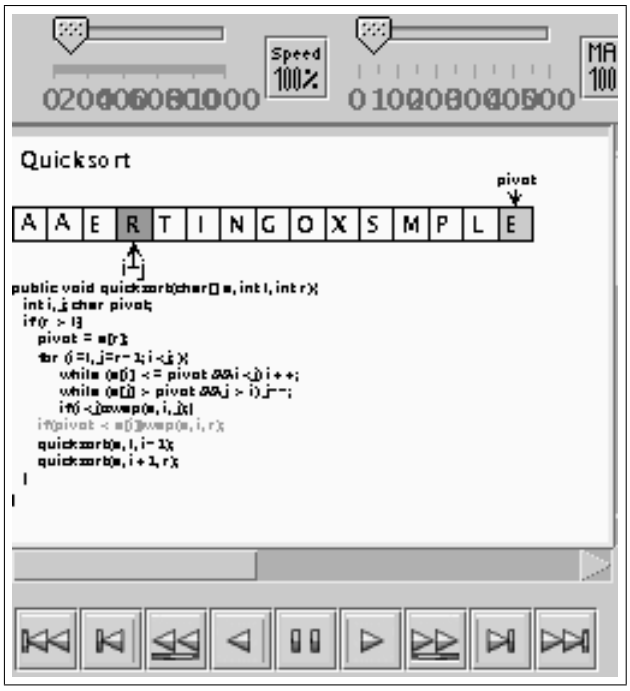

Figure 1: ANIMAL animation example, scaled to 41%

4 Developed System

Our combined system uses JHAVÉ as the client/server component in which the animations to be viewed are selected. Once a particular animation is chosen, JHAVÉ executes a program on the server that produces a script for that algorithm. The client then sends *readScript* and *runScript* messages to the appropriate *Visualizer* to parse and render this script.

Figure 1 shows a Quicksort animation displayed using the ANIMAL *Visualizer*. The artifacts are introduced by scaling the display to fit the paper layout. The top of the window contains sliders controls for the animation display speed and magnification. The range of the settings is 0% to 1000% for speed and 0% to 500% for magnification. Two buttons for resetting the values to 100% are also provided. The bottom toolbar contains the animation controls. The animation can be displayed smoothly or using discrete steps, as well as in a "slideshow" mode. The latter links subsequent animation steps with a user-defined delay. All controls work both in forward and reverse mode. Finally, the user can also jump to the start or end of the animation and pause the display. Thus, *rewinding* and *smooth motions* are fully supported by the engine.

Figure 2 shows an accompanying window with an example of a stop-and-think question. If this question is presented as part of a "for-real" quiz, ANIMAL will freeze the forward-directed animation controls during a quiz, so that the student cannot simply step forward to find the answer. A separate "time line" window offers a structured view of the animation, as required in Section 2. The animation author can select the steps she regards as important for this view. A click on an entry updates the animation to the associated step.

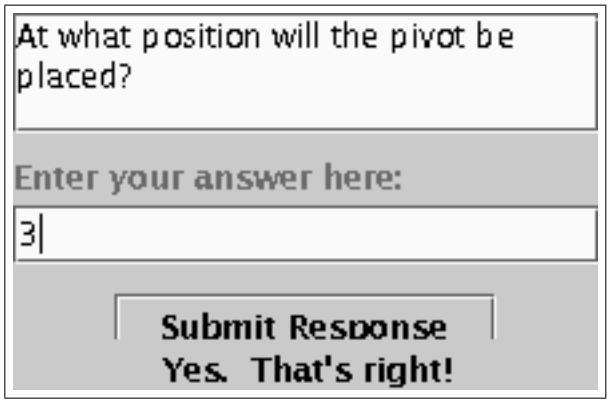

Figure 2: Example stop-and-think question window

Within the course of the cooperation, ANIMAL was also extended to open external HTML-based documentation from a URL specified in the script read from the server. Additionally, ANIMAL extensions offer diverse export formats including various image formats such as *JPG* or *PNG*, as well as *Quicktime* videos. Due to native code implementation employed in the API, the latter is currently restricted to Windows and MacOS. Finally, the graphical user interface of ANIMAL can be translated by a single mouse click. If the animation author has coded the animation according to our internationalization specification, the text components of the animation can also be translated on loading, adapting the po-

sition of other animation objects relative to the text.

5 Conclusions

The stage has been set to begin a much different emphasis in AV. Hundhausen states "*how* students use AV technology has a much greater impact than *what* AV technology shows them" [7]. Bigger and better graphics are not necessarily the answer to effective learning. More importantly, AV systems must provide instructors with a collection of tools that make it relatively painless to incorporate AV into their courses and then evaluate its effectiveness in an empirical fashion – similar to the way we have evaluated textbooks in the past. By offering a far-reaching set of pedagogical features, ANIMAL in JHAVÉ represents a first step in this direction.

The JHAVÉ server presently produces GAIGS or JSamba scripts with accompanying hypertext materials and stop-and-think questions for twenty-four algorithms covering such diverse topics as hashing, sorting, string-searching, graph and tree algorithms. We are in the process of adding programs to produce a parallel set of ANIMAL animations. When completed, this work will provide a rich testbed to compare and contrast the effectiveness of various types and modes of algorithm visualization.

References

[1] Anderson, J. M., and Naps, T. L. A Context for the Assessment of Algorithm Visualization System as Pedagogical Tools. *First International Program Visualization Workshop, Porvoo, Finland. University of Joensuu Press* (July 2001), 121–130.

[2] Baecker, R. *Sorting Out Sorting*: A Case Study of Software Visualization for Teaching Computer Science. In *Software Visualization*, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds. MIT Press, 1998, pp. 369–381.

[3] Bazik, J., Tamassia, R., Reiss, S. P., and van Dam, A. Software Visualization in Teaching at Brown University. In *Software Visualization*, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds. MIT Press, 1998, ch. 25, pp. 382–398.

[4] Bonwell, C. C., and Eisen, J. A. Active Learning: Creating Excitement in the Classroom. Tech. rep., George Washington University, Washington, DC, 1991.

[5] Gloor, P. A. User Interface Issues For Algorithm Animation. In *Software Visualization*, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds. MIT Press, 1998, ch. 11, pp. 145–152.

[6] Grissom, S., and Naps, T. Yet Another Experiment Using Algorithm Visualization to Teach Computer Science. *Submitted for the 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002), Århus, Denmark* (2002).

[7] Hundhausen, C. D. A Meta-Study of Software Visualization Effetiveness. WWW: `http://lilt.ics.hawaii.edu/~hundhaus/writings/`, 1997.

[8] Jarc, D., Feldman, M. B., and Heller, R. S. Assessing the Benefits of Interactive Prediction Using Web-based Algorithm Animation Courseware. 31st *SIGCSE Technical Symposium on Computer Science Education, Austin, Texas* (Mar. 2000), 377–381.

[9] Naps, T., Eagan, J., and Norton, L. JHAVÉ: An Environment to Actively Engage Students in Web-based Algorithm Visualizations. 31st *SIGCSE Technical Symposium on Computer Science Education, Austin, Texas* (Mar. 2000), 109–113.

[10] Naps, T. L. Incorporating Algorithm Visualization into Educational Theory: A Challenge for the Future. *Informatik / Informatique Special Issue on Visualization of Software* (Apr. 2001), 17–21.

[11] Petre, M., Blackwell, A., and Green, T. Cognitive Questions in Software Visualization. In *Software Visualization*, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds. MIT Press, 1998, ch. 30, pp. 453–480.

[12] Pierson, W., and Rodger, S. H. Web-based Animation of Data Structures Using JAWAA. 29th *SIGCSE Technical Symposium on Computer Science Education, Atlanta, Georgia* (1998), 267–271.

[13] Rößling, G., Schüler, M., and Freisleben, B. The ANIMAL Algorithm Animation Tool. *5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000), Helsinki, Finland* (July 2000), 37–40.

[14] Stasko, J. Samba Algorithm Animation System, 1998. Available at `http://www.cc.gatech.edu/gvu/softviz/algoanim/samba.html`.

[15] Stasko, J. Smooth Continuous Animation for Portraying Algorithms and Processes. In *Software Visualization*, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds. MIT Press, 1998, ch. 8, pp. 103–118.

[16] Stasko, J., and Lawrence, A. Empirically Assessing Algorithm Animations as Learning Aids. In *Software Visualization*, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds. MIT Press, 1998, ch. 28, pp. 419–438.

[17] Stern, L., Søndergaard, H., and Naish, L. A Strategy for Managing Content Complexity in Algorithm Animation. 4th *Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE'99), Cracow, Poland* (Sept. 1999), 127–130.