Algorithm Animation Revisited

Mark Fienup and Daniel Tesfa
Computer Science Department
University of Northern Iowa
219 Wright Hall
Cedar Falls, Iowa 50614-0507
fienup@cs.uni.edu and dant@distek.com

Abstract

Algorithms are inherently difficult for students to learn due to the abstractness of (1) the basic building blocks such as arrays or other data structures, and (2) the algorithmic technique underlying the algorithm such as divide-and-conquer, dynamic programming, etc.

Algorithm animation seems like an intuitively useful pedagogical tool for explaining how algorithms work, since a graphical visualization of these abstractions should make them more concrete. Not surprisingly, several algorithm animations systems have been developed during the last several decades to help teach computer science concepts.

Unfortunately, experimental studies designed to evaluate the effectiveness of algorithm animation have shown that visualization itself is ineffective at improving learning. This paper summarizes this body of work for suggestioned best practices to improve the effectiveness of algorithm animation as a learning tool. Additionally, we survey existing Algorithm animation tools using these best practices as a guideline for evaluation.

1 Introduction

Algorithms (and programming in general) are inherently abstract due to the abstractness of:

1. the basic building blocks such as arrays or other data structures, and
2. the algorithmic technique underlying the algorithm such as divide-and-conquer, dynamic programming, etc.

Intuitively, algorithm animation seems like a useful pedagogic tool for explaining how algorithms work, since a graphical visualization of these abstractions should make them more concrete. Not surprisingly, several algorithm animations systems have been developed during the last several decades to help teach computer science concepts. In fact, animations produced by more recent systems are available for most common algorithms via the web as applets (e.g., http://www.cs.hope.edu/~alganim/ccaa/).

Unfortunately, the algorithm animation tools used to produce these algorithms are nontrivial to learning. Instructors wanting to animate a specify algorithm in their textbook may face too much overhead in time and effort to make visualization worthwhile. Ideally, these tools would be simple enough to use, so that a student with an incorrect sorting algorithm could animate it easily and gain some insight into where their code is wrong.

Experimental studies designed to evaluate the effectiveness of algorithm animation have shown that visualization itself is ineffective at improving learning (Hundhausen, Sarah Douglas, and John Stasko, 2002). However, the ITiCSE working group on "Improving the Educational Impact of Algorithm Visualization" believes that algorithm animation can improve learning if it engages learners in an active learning activity, and they summarizes eleven best practices for doing this (Naps et al., 2003). The first part of this paper offers suggestions for instructors currently using algorithm animation even if they are just tracing a sorting algorithm at the blackboard.

In the second half of this paper, we survey existing algorithm animation tools using these best practices as a guideline for evaluation.

2 Best Practices of Algorithm Animation

Hundhausen, Douglas, and Stasko (2002) reviewed and analyzed the findings of twenty-four experimental studies on the effectiveness of algorithm visualization in learning. They concluded that when students only view an algorithm's visualization, then they did not learn significantly more than students who used conventional learning materials. However, if algorithm visualization technology is used to actively engage students when viewing the animation, then learning can be enhanced. Activities used to actively engage students were:

- what-if analyses of algorithmic behavior

- prediction exercises
- programming exercises

They hypothesized via constructivist learning theory that these actives helped enable students to construct their own understanding of the algorithms.

Later work by the ITiCSE 2003 Working Group on Improving the Educational Impact of Algorithm Visualization (Naps et al., 2003) concluded that the two key obstacles to visualization technology's widespread usage were:

1. learners may not find the visualization technology educationally beneficial, and
2. instructors may incur too much overhead developing the visualization to make it worthwhile.

They agreed that visualization technology is of little educational value unless it engages learners in some type of active learning activity. As part of their paper, they summarizes eleven best practices for algorithm animation. Table 1 contains these best practices.

Best Practice		Suggested Techniques
1	Provide resources that help learners interpret the graphical representations and their relation to program elements.	Explain the relationship by embedding the representations in the system using text or narration.
		Reinforce the relationship by allocating instruction time to the topic during the course.
2	Adapt to the knowledge level of the user.	Novices benefit by a simple interface, preselected animation data, and possibly animations based on well-know metaphors such as comic strips, slide shows, etc.
		Advanced learners benefit by being able to input animation data.
3	Provide multiple views of the animation.	It is good to couple a view of the code and view the state of the data structure with a more abstract view of the animation.
4	Include performance information.	Include to aid understanding of an algorithms efficiency which is an important part of understanding an algorithm,
5	Include execution history.	Allow the learner to view previous algorithmic steps to refresh their memory and clear up misconceptions about previous steps.

Table 1: Best Practices for Algorithm Animation

	Best Practice	Suggested Techniques
6	Support flexible execution control.	Allow the learn to view the algorithm animation both forwards and backwards with simple video player like controls.
7	Support learner-built visualizations.	If the learner develops the visualization, they gain insight into the algorithm and have a greater sense of responsibility for the visualization.
8	Support custom input data sets.	Gets the learner more actively involved in the animation and allows them to more fully exercise algorithm features.
9	Support dynamic questions.	Periodically ask short answer questions to focus a learners attention and promote self-examination to improve comprehension.
10	Support dynamic feedback.	If a learner is able to make predictions about the future steps of the animation, then provide feedback on their performance.
11	Complement visualization with explanations.	Provide coordinated explanation of the animation either textually, or via audio information.

Table 1 (continued): Best Practices for Algorithm Animation

As firm believers of active learning in the classroom, these best practices for algorithm animation have strong implications about effective classroom instruction as well. For example, a CS 1 instructor could show the code for a sorting algorithm on one PowerPoint slide, and follow it on the next slide with a complete trace of its execution by showing static snapshots of changes made to an array. While this might provide multiple views of the animation. It does not couple a view of the code simultaneously with a view of the state of the data structure. Additionally, by showing the complete trace of the sort, the instructor lost the opportunity to dynamically question the students about the future state of the array during the next iteration of the outer-loop.

Tudoreanu (2003) studied the effect of "cognitive economy," i. e., the economy of information and tasks related to the visualization. He concluded that for an animation and its animation environment to positively impact learning, it is important to reduce the amount of data and tasks required in the visualization session. Thus, allowing the user to focus on the algorithm.

In a broader context, animation as a tool for instruction has been studied by psychologists. Tversky, Morrison, and Betrancourt (2002) tried to answer the question whether

animation is more effective than static graphics for learning. This was a meta-study of previous experimental studies. They suggest two necessary, but not sufficient conditions for a successful animation: the congruence principle and the apprehension principle. The congruence principle states that the structure and content of the external representation should correspond to the desired structure and content of the internal representation. Since animation represents change over time, there should be a natural correspondence between change over time and the essential conceptual information being conveyed. The apprehension principle states that the structure and content of the external representation should be readily and accurately perceived and comprehended. Animations are often too complex or too fast to be accurately perceived. Clearly, the apprehension principle corresponds well with Tudoreanu's (2003) cognitive economy concept.

For algorithm animations, we conjecture that the temporal sequence of instruction execution and its corresponding effect on the algorithms data structure(s) satisfy the congruence principle. Taking the apprehension principle into account, algorithm animations must be slow and clear enough for students to perceive the correlation between the execution of programming statements and changes to the data structure(s). Highlighting to focus attention on this correlation should improve the effectiveness of the algorithm animation.

When trying to answer the question whether animation is more effective than static graphics for learning Tversky, Morrison, and Betrancourt (2002) where careful to throw out studies that not only added animation, but inadvertently included other factors that are know to facilitate learning such as interactivity or prediction. Since our goal is to facilitate learning, inclusion of interactivity and prediction in an algorithm animation would be a good thing.

3 Algorithm Animation Tools

In the second half of this paper, we survey existing algorithm animation tools by categorizing them by functionality. For each category, we evaluate their ability to overcome the two key obstacles to visualization technology identified by the ITiCSE 2003 Working Group on Improving the Educational Impact of Algorithm Visualization (Naps et al., 2003), i. e.,

1. their ability to be educationally beneficial for learners, and
2. their ability to allow instructors (or students) to developing worthwhile visualizations without incurring too much overhead.

For the assessment of this first criteria we will use the best practices outlined earlier as our guide.

3.1 Scripting Languages for Algorithm Animation Engines

Several algorithm-animation scripting languages such as XTANGO, SAMBA, JAWAA, etc. have been developed that allow users to develop algorithm animations. These types

of animation consists of creating a script file that contains low-level commands for the construction of graphical objects and their motion within a window or applet. Figure 1 shows a sample JAWAA script that animates a breadth-first search (BFS) of a graph.

```
#begin
  node 1 150 30 20 20 1 1 black lightGray black CIRCLE
  node 2 100 120 20 20 1 2 black lightGray black CIRCLE
  node 3 200 120 20 20 1 3 black lightGray black CIRCLE
  node 4 150 200 20 20 1 4 black lightGray black CIRCLE
  node 5 250 200 20 20 1 5  black lightGray black CIRCLE
  node 6 200 250 20 20 1 6 black lightGray black CIRCLE
  node 7 50 200 20 20 1 7 black lightGray black CIRCLE
  node 8 100 250 20 20 1 8 black lightGray black CIRCLE
#end
#begin
  connectNodes 9 1 3 black 1
  connectNodes 10 1 2 black 1
  connectNodes 11 3 4 black 1
  connectNodes 12 3 5 black 1
  connectNodes 13 4 5 black 1
  connectNodes 14 4 2 black 1
  connectNodes 15 4 8 black 1
  connectNodes 16 7 2 black 1
  connectNodes 17 4 6 black 1
  connectNodes 18 6 5 black 1
  connectNodes 19 8 7 black 1
#end
text t1 250 30 "L0: " blue
text t2 250 50 "L1: " red
text t3 250 70 "L2: " orange
text t4 250 90 "L3: " magenta
text t5 250 110 "L4: " cyan
changeParam 1 bkgrd green
delay 600
marker 20  1  10 black green
moveMarker 20 2
changeParam 2 bkgrd green
marker 21  1  10 black green
moveMarker 21 3
changeParam 3 bkgrd green
changeParam 1 bkgrd blue
delete 20
text t6 280 30 "1 " blue
moveMarker 21 4
changeParam 4 bkgrd green
marker 22 3 10 black green
moveMarker 22 5
changeParam 5 bkgrd  green
delete 22
changeParam 2 bkgrd  red
changeParam 3 bkgrd  red
text t7 280 50 "2 3 " red
moveMarker 21 8
changeParam 8 bkgrd  green
marker 22  4  10  black  green
moveMarker 22 6
changeParam 6 bkgrd  green
delete 22
changeParam 4 bkgrd  orange
delay 500
changeParam 5 bkgrd  orange
text t8 280 70 "4 5 " orange
moveMarker 21 7
changeParam 7 bkgrd  green
changeParam 8 bkgrd  magenta
changeParam 6 bkgrd  magenta
text t9 280 90 "8 6 " magenta
delete 21
changeParam 7 bkgrd cyan
text t10 280 110 "7 " cyan
```

Figure 1. JAWAA script for breadth-first search of a graph

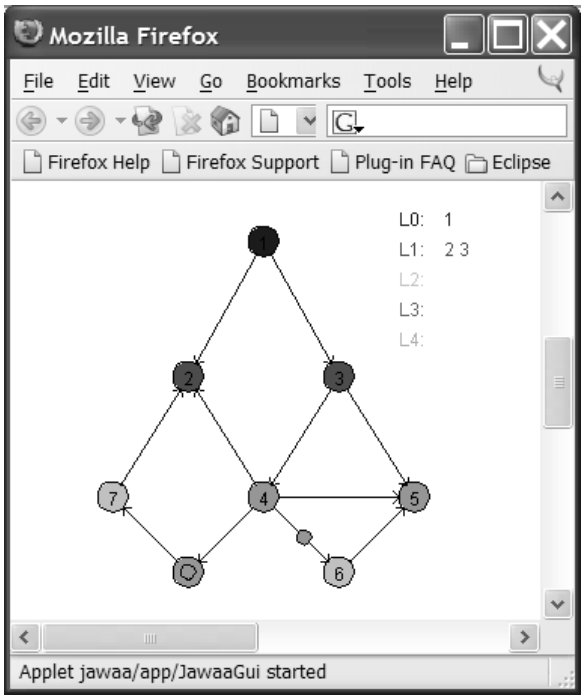

Figure 2. JAWAA applet showing breadth-first search.

Figure 2 shows the "execution" of the applet generated by the JAWAA script in Figure 1 to the point of excuting the command "***moveMarker 22 6***". Excerpts from the JAWAA website (URL: http://www.cs.duke.edu/csed/jawaa2/commands.html) that describe several of the JAWAA commands used in the script are listed below in Figure 3.

node [name] [x] [y] [width] [height] [n] [data..] [color] [bkgrd] [textcolor] [type]
Creates a stand alone node with the specified attributes and data items. Type refers to the graphic representation of the node, either CIRCLE or RECT.

connectNodes [name] [node1] [node2] [color] [num]
This command creates an arrow connection from one node to another. The arrow that is drawn can be referenced by the specified name. The arrow has the specified color and 0, 1 or 2 arrowheads as specified by the num parameter.

marker [name] [node] [diameter] [color] [bkgrd]
Creates a marker that is located at the specified node with the defined attributes.

moveMarker [target] [node]
This command can only be used when the target object is a marker. The marker is moved to the new node.

Figure 3. Description of some JAWAA commands

The advantages of these algorithm-animation scripting languages are:

- the scripting commands are relatively straightforward and easy to learn and use
- a program written in any language can be animated by inserting print statements that generate the scripting commands
- the scripts are extremely flexible since the script writer has complete control over the creation and movement of the graphical objects
- a simple ASCII editor is needed to write scripts
- the scripting objects can be tailored for algorithm animation, e.g., JAWAA graphical objects include arrays, linked lists, queues, and stacks

Overall, these scripting languages come close to allowing instructors (or students) to develop worthwhile visualizations without incurring too much overhead. Their biggest drawback is the low-level nature of the scripting commands, e.g., you need to specify the x, y coordinates of objects. However, they are relatively simple to understand and would support the practice of learner-built visualizations.

Unfortunately, these algorithm-animation scripting languages also have the following serious drawbacks with respect to their ability to be educationally beneficial for learners:

- they do not provide resources that help learners interpret the graphical representations and their relation to program elements
- they do not adapt to the knowledge level of the user
- they do not provide multiple views of the animation
- they do not support flexible execution control
- they do not support dynamic questions
- they do not support dynamic feedback
- they do not complement visualization with explanations

The lack of interactive support and the decoupling of algorithm animation from program code are serious problems. With careful design, it might be possible to build into the animation script performance information, a static trace of the execution history, and support for custom input data sets.

3.2 Interactive Debuggers with Data Structure Visualization

The Lens system (Mukherjea and Stasko, 1994) was an early attempt at combining algorithm animation support with an interactive debugger. It incorporated a graphical tool kit within a debugger to aid in the development of animation views by allowing the user to associate animation actions (e.g., create graphical object, move object, color object, flash object, etc.) with specific lines of source code. Additionally, Lens provided "templates" for visualizing common data structures such as arrays, link lists, and binary trees. When the user runs the program inside Lens, it pops up an animation window. If the animation does not sufficiently show what the programmer intended, the animation commands can be modified accordingly.

While the Lens system is interactive and couples the algorithm's source code and its execution with animation views, it makes a better algorithm animation prototyping tool

for instructors than an educationally beneficial tool for student learners. A student attempting to understand or develop an algorithm using Lens will not know if sources of confusion are from an erroneous program or erroneous animation actions.

The jGRASP IDE (version 1.8.0 Beta) by Hendrix, Cross II, and Barowski (2004) provides automatic visualization of Java arrays and Java collections classes inside the debugger. It does not provide true algorithm animation, but allows self-paced visualization of these data structures as you step through the debugger without the need to generate the animation.

Along these same lines, JIVE (Java Interactive Visualization Environment) by Gestwicki and Jayaraman (2004) is a powerful tool for visualizing object structure and execution histories. Not only should it be able to visualize commonly used data structures, but it allows for both forward and backward execution stepping. It provides for multiple views of object states in different granularities, so novices can see less details and experts can dive to a deeper level.

3.3 Questioning-Based Algorithm Animation Tools

The JHAVE (Java-hosted Algorithm Visualization Environment) tool by Naps, Eagan, and Norton (2000) is a client-server architecture that allows instructors to provide students algorithm animations in both a smooth and discrete visualization of the animation. The animation is coordinated with textual information to provide further explanation. To force students to actively participate in the visualization, the animation is periodically stopped and students are asked "stop-and-think" questions to predict the next step, etc. The level of involvement by the student while viewing the animation helps make for an educationally beneficial, interactive experience for the student.

4. Conclusions

No single algorithm animation tool is currently sufficient to overcome both of the key obstacles of algorithm visualization:

1. their ability to be educationally beneficial for learners, and
2. their ability to allow instructors (or students) to develop worthwhile visualizations without incurring too much overhead.

However, several tools do a reasonably good job at overcoming a single obstacle. Algorithm-animation scripting systems, while tedious, are relatively simple to use, but if the resulting animation is passively watched by students, then they appear to have little educational benefit. Animation systems, such as JHAVE, that actively engage the user are a step in the right direction. JIVE is probably the best overall algorithm animation tool currently available since it provides "automatic" visualization of commonly used data structures, and forward and backward execution stepping. Hopefully, future

algorithm animation systems can combine the best traits of these systems and overcome both of the key obstacles to algorithm animation.

References:

Peter Brummund, current update by Ngozi V. Uti, "Complete Collection of Algorithm Animations (CCAA)" website at http://www.cs.hope.edu/~alganim/ccaa/

T. Dean Hendrix, James H. Cross II, and Larry A. Barowski, "An Extensible Framework for Providing Dynamic Data Structure Visualizations in a Lightweight IDE", 35th ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2004), Norfork, Virginia, March 2004, Austin, Texas, pp. 387-391.

Christopher Hundhausen, Sarah Douglas, and John Stasko, "A Meta-Study of Algorithm Visualization Effectiveness", Journal of Visual Languages and Computing, Vol. 13, No. 3, June 2002, pp. 259-290.

Sougata Mukherjea and John T. Stasko, "Toward visual debugging: integrating algorithm animation capabilities within a source-level debugger", ACM Transactions on Computer-Human Interaction, Vol. 1, Issue 3, September 1994, pp. 215-244.

Thomas Naps, James Eagan, and Laura Norton, "JHAVE -- An Environment to Actively Engage Students in Web-based Algorithm Visualizations", 31st ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2000), March 2000, Austin, Texas, pp. 109-113.

Thomas L. Naps, Guido Robling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. Angel Velazquez-Iturbide, "Exploring the role of visualization and engagement in computer science education", ACM SIGCSE Bulletin, Vol. 35, Issue 2, June 2003, pp. 131-152.

M. Eduard Tudoreanu, "Designing effective program visualization tools for reducing user's cognitive effort", Proceedings of the 2003 ACM Symposium on Software Visualization, June 11-13, 2003, San Diego, California, pp. 105-114.