Designing Effective Algorithm Visualizations

Sami Khuri[*]
Department of Mathematics and Computer Science
San José State University
San José, CA 95192, USA
E-mail: khuri@cs.sjsu.edu
WWW: http://www.mathcs.sjsu.edu/faculty/khuri

Abstract

Advances in computing technology and the affordability of software and high-performance graphics hardware enabled rapid growth of visual tools. Today, not only very expensive workstations, but also low-cost PCs are capable of running computationally demanding visualization systems. Algorithm visualizations or the graphic depiction of algorithms in execution have been used in explaining, designing and analyzing algorithms since the early 1980s. Although many tools have been developed over the past twenty years, not enough attention has been paid to the analysis of users, their needs, tasks, and goals. This paper provides an overview of algorithm visualization techniques, based on the premise that a rethinking of algorithm animation design is required in order to harness its power for enhancing learning. More information about the topic can be found on the author's web page.

1 Introduction

This paper provides an overview of algorithm visualization techniques, based on the premise that a rethinking of algorithm animation design is required in order to harness its power to enhance learning. Since the early 1980s, many visualization systems have been created. They range from animations of one algorithm or a group of related algorithms, such as MLFQ, PAGE (Khuri & Hsu, 1999), RLE, Quadtree, and JPEG (Khuri & Hsu, 2000), to powerful distributed and collaborative algorithm animation systems, such as Fuse-N

[*]Part of this work was done while the author was on sabbatical leave at the University of Málaga, 29071 Málaga, Spain.

by MIT Computer Graphics Group (Teller, Boyd, Porter, & Tornow, 1998) and JCAT (Brown, Najork, & Raisamo, 1997). Algorithm visualizations can be used to attract students' attention during lecture, explain concepts in visual terms, automate examples and demos and encourage a practical learning process. They facilitate better communication among the students and the instructors. The instructional process in traditional classes is often one-directional. Instructors use the whiteboard or slides to present the material. The interaction between them and the students is limited to verbal discussions. Very rarely do students have the opportunity to experiment in class and explore a concept in other dimensions. Interactive algorithm visualizations provide new opportunities for instruction. The instruction approach can now be viewed as a combined learning process. Students do not just sit in the classroom and listen. They use their left and right brain to visualize things happening while they are processing their mental thoughts. They can learn by doing and do not have to worry about making mistakes.

This paper focuses on important steps of the algorithm visualization design and techniques of effective educational illustrations of algorithms.

The rest of the paper is organized as follows. Section 2 discusses some important issues often overlooked by the developers of algorithm visualization. Section 3 presents an overview of techniques that can make visualizations more effective. Section 4 concludes the paper with some recommendations for designers of new visualizations.

2 Design Issues

Like many other design disciplines, a successful algorithm visualization design should consider effective representation and presentation of information, such as layout, color, graphics, and user interface. Designers of interactive algorithm visualizations must be able to blend a thorough knowledge of technical feasibility with a mystical aesthetic sense of what attracts users.

Some issues that sometimes have been overlooked by the designers of algorithm visualizations include the analysis of the users, their needs, their tasks, the scope of the package, and the resources available to developers. The analysis should be done as a preliminary step of the design process. What follows are examples of how it can influence the resulting algorithm visualization.

- Although some systems are intended for a wide audience, no algorithm visualization will ever be universally superior across all kinds of users.

Understanding who the users are should determine the content, organization, breadth, depth, access and presentation methods of the visualization system being designed.

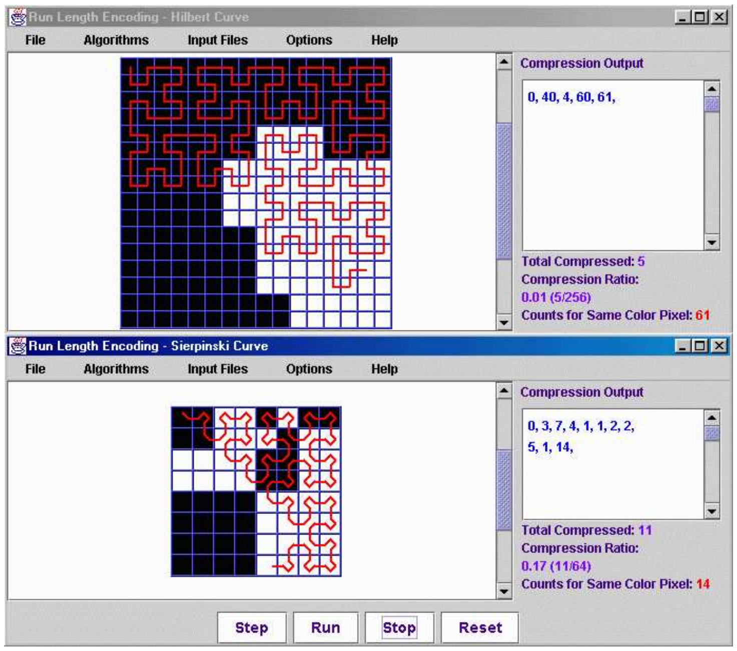
For example, novice users need help in mapping real world models onto a program. For this type of users, visualization should include a number of worked examples showing how the algorithm can be used and how the algorithm output is related to its input, Help files should be available to describe how the algorithm works, as well as how the interface is organized. Short quizzes should be provided for students to make sure they have understood the material, and to exercise the use of the visualization tools.

On the other hand, experienced users will want a system that will allow them to move easily between the code and the visualization, or to integrate their own programs into the system. The methods for calling visualization routines should be efficient and well documented.

- When designing an algorithm visualization system, it is important to note the system's intended goals and select the content accordingly. An assessment should be made to see if presenting an algorithm by visualizing it is the most effective way. For example, using a telephone book to explain the binary search algorithm is probably more efficient than implementing an algorithm visualization.

- Algorithm visualization systems have been used to construct visualizations for lecture demonstrations (Brown, 1988), as the basis for interactive labs (Naps, 1990), or as visualization assignments in which students construct their own visualizations of the algorithms under study (Stasko, 1997). In more recent work, Hundhausen proposes to use software visualization in one of the following situations: lectures, assignments, laboratories, study, office hours, and tests (Hundhausen, 1999). Obviously, each situation demands a different kind of animation system, and it is difficult to build the system that satisfies all of them.

  For example, if the tools are developed primarily for classroom demonstration with an instructor describing the scenario, they do not need a lot of written explanations. As a visualization proceeds, the explanation of what is happening should be left for the instructor and not explicitly described in the demonstration. If, however, the tools will be used for self-directed study, they will need a lot of explanatory cues in the form of short textual notes.



```
Compression Output
0, 40, 4, 60, 61,
Total Compressed: 5
Compression Ratio:
0.01 (5/256)
Counts for Same Color Pixel: 61
```

```
Compression Output
0, 3, 7, 4, 1, 1, 2, 2,
5, 1, 14,
Total Compressed: 11
Compression Ratio:
0.17 (11/64)
Counts for Same Color Pixel: 14
```

Figure 1: Compressing bitmaps using the RLE package.

- One of the most difficult parts of the analysis step is to decide which information should the visualization convey and how to present it. Existing systems have attempted to visualize data structures, program flows, pseudocode and the algorithm in action.

  Designers should develop an appropriate set of conventions to denote different information, so that the users do not waste their time trying to figure out what the picture means. The information should be carefully abstracted. Different levels of abstraction require different representation methods (Cox & Roman, 1992). Direct representations directly map information to the display, e.g. the bitmap in Figure 1 can be easily reconstructed from the picture in the left panel. In structural representations, some details of information are concealed and the remaining information is directly represented. For example, proportionally-sized color blocks may indicate memory allocation and usage without attempting to present the state of memory. Synthesized representations, such as "Compression Output" in Figure 1, can be derived from the program data, but is not directly represented in the program. The extra data structures are created and their contents are periodically examined and updated in the display. This type, as well as explanatory representations are added in order to improve understandability of the display and focus the attention of the viewers.

Information about the algorithm can be represented through shape, size, color, texture, sound, and arrangement of objects. Different, but related algorithms can be animated using the same representation or different representations for each algorithm. Using the same representation is productive since once the animation view has been established for the first algorithm, the view can then be reused. This approach also eases the comparison of the behavior of related algorithms as in "Sorting Out Sorting" (Baecker, 1998).

- The designer also has to set the boundaries for the visualization. Most modern algorithm visualizations allow the user to enter their own data sets. If the input file contains a lot of data, the visualization will become very complex and difficult to see. The users will have to scroll and may lose the sense of the "whole" picture. The animation might confuse more than educate. In such cases, visualization should condense complicated parts of the scene into smaller items. Algorithm's execution can also be condensed, e.g. several phases of an algorithm can be omitted and only the final result of those steps is presented.

3 Some Techniques for Creating Visualizations

In this section, we examine techniques for visual representation of information about programs. We focus on approaches that are either fundamental or have been tried and appear to enhance visual communication. Although research on algorithm visualization dates back to 1980s, no definitive lexicon of these techniques exists. In addition, new techniques continually evolve. We thus try to give a reasonably complete coverage of the area, bearing in mind, that it cannot be comprehensive.

3.1 Display Layout

The screen on which the visualization occurs can easily get cluttered with the many visual representations of control constructs or data items. If these are reduced in size to allow yet more representations to be displayed then the animations are too small to be seen clearly. One way to reduce visual cluttering is to divide each animation display into *functional areas*, each containing a different type of information. It is advisable to place important information near the top and to the left (eye-motion studies show that our gaze goes to the upper-left of a rectangular display and then moves clockwise). Each type of information should be consistently displayed in its assigned area. If necessary, areas can be enlarged or minimized to handle special cases, always keeping the primary viewing area as large as possible.

It is often useful to provide multiple views of the same system in order to understand a variety of characteristics of the data. Multiple views might include simultaneous coarse-grained and fine-grained views of data structures, or a graphical view of the changing program data with a corresponding view of the executing source code (Brown, 1988). One of the advantages of multiple views is their ability to avoid forcing the viewer to remember algorithm states no longer on display. For example, two consecutive frames are shown in the "comic strip" approach (Biermann & Cole, 1999). These frames display the state of the system both immediately before and after an action. A useful extension of the multiple views idea is the use of segmentation, where a selection of some subset of the nodes in one view is reflected in all views. In this case, one could select a physical region in one of the views, and the display would immediately highlight the corresponding values in the other views (Khuri & Hsu, 2000).

Some researchers criticize the multiple views approach. They argue that multiple views lead to confusion about what is being explained. They believe that a single window should be used to display the animation with the explanatory text, thus preventing problems arising from too much information being displayed on the limited resolution device.

3.2 Using Color

Color has traditionally been used to enhance black-and-white information. With respect to learning and comprehension, color is superior to black-and-white in terms of processing time and the viewer's emotional reactions. But research has shown that there is no difference in the viewer's ability to interpret information: people do not learn more from a color display, though they may say they do. The crucial factor is that color is more enjoyable and easier to remember.

Establishing general rules or specifications for color use is difficult. We still lack some important understanding of color vision. Some general guidelines have appeared in computer graphics magazines. What follows is a short summary of these guidelines.

1. With respect to color, it is best to be conservative. Use a maximum

of five, plus or minus two, colors. For novice viewers, four distinct colors are appropriate. This allows extra room in short-term memory (lasting about 20 seconds), which can store five words or shapes, six letters, seven colors and eight digits.

2. Use foveal (center) and peripheral colors appropriately. For example, use blue for large areas, such as backgrounds and not for text, thin lines or small shapes. Blue-sensitive cones are the least numerous color receptors in the retina, and the eye's central focusing area, the fovea, contains a relatively small number of these cones.

   Use red and green in the center of the visual field, not in the periphery. The edges of the retina are not particularly sensitive to these colors. If they are used at the periphery, some signal to the viewer must be given to capture the user's attention - for example, size change, blinking, etc.

3. Do not use high-chroma, spectrally extreme colors simultaneously. Strong contrasts of red/green, blue/yellow, green/blue and red/blue create vibrations, illusions of shadows and afterimages.

4. Use familiar, consistent color codings with appropriate references. Some common Western denotations are:

   - Red refers to stop, danger, hot, fire.
   - Yellow refers to caution, slow, test.
   - Green refers to go, O.K., clear.

5. Be consistent in using the same color for grouping related elements. Do not use a particular color for elements not related to the others, such as data structure and control buttons. Similar background colors of related areas can orient the viewer to understand the conceptual linking of the two areas, without the need of more explicit verbal cues.

6. Use high value, high saturation colors to draw attention. The use of bright colors for danger signals, reminders, etc., is appropriate. High chroma red or blue alerts seem to elicit faster responses than does yellow or yellow-orange.

7. Use color to save screen area. For example, using a small area changing in color to denote a progress, rather than a bar or line, can greatly economize space.

3.3 Using Sound

Sound is a useful complement to the visual output because it can increase (or reduce) the amount of information communicated to the user. It makes use of the auditory system which is powerful but underutilized in most current interfaces. So far, audio has been used to improve the programmer's awareness of the behavior of parallel programs by generating sounds based on trace data recorded during execution (Jackson & Francioni, 1992). While animating algorithms, Brown and Hershberger generated sounds corresponding in pitch to elements being inserted into a hash table, items being sorted, and to the number of active threads (Brown & Hershberger, 1992). The following are some recommendations for using sound in algorithm visualization systems.

- Users have their own distinct preferences for non-speech sounds while they are learning and using the system, so a highly-flexible user-configuration component should be included for the users to specify:
  - the exact sound for each event/command,
  - whether each specific sound is on/off,
  - whether all the sounds are on/off.

  This configuration should be easily accessible.
- The use of any non-speech sound is likely to require some training and practice. A mixture of documentation and on-line training with examples could be used to give users the correct interpretation for the sounds in a system. This will help to ensure that users have the intended model for the sounds.
- Carefully consider the number of sounds. Use only a few sounds for the most important or difficult events/commands. Don't create a system full of non-speech sounds, which might become irritating. In addition, there is a threshold for learning and remembering sounds (from 7 to 9 different sounds).

3.4 Selecting Input Data

The complexity of a visual presentation is generally proportional to the amount of information being conveyed. Therefore, it is better to launch a presentation with a relatively small problem instance and provide larger ones for the users who begin to understand the meaning associated with the visual patterns unfolding on the screen. The RLE package (see Figure 1) can visualize algorithms for input files of size 4x4, 8x8, 16x16, and 32x32. Input files larger than that will be difficult to see on the screen, and thus, not beneficial to the students learning the algorithms. For pedagogical purposes, pathological cases should also be provided (e.g. all-white and all-black bitmaps for run length encoding algorithms). Graphical input tools are becoming a must, especially if visualization is intended for free exploration by the students. When designing input tools, possible erroneous situations should be considered. For the package in Figure 1, the erroneous input would be a text or an executable file for example. Large amounts of data are needed to compare the performance of different algorithms. Another useful technique for comparing algorithms is to have them run side-by-side. BALSA animations of sorting algorithms effectively use this technique (Brown, 1988).

3.5 Providing Interactivity

Probably the most important issue in designing algorithm visualizations is that of user interaction. Interaction is what distinguishes algorithm visualization systems from the simple movie demonstration of algorithms. The degree, methods, and forms of interactivity will depend on the tasks users want to accomplish. If the system will be used for exploratory purposes, like Brown's University Exploratories Project (Simpson, Spalter, & Dam, 1999), users' interaction will be dynamic, frequent, and sometimes unpredictable. Algorithm visualizations should encourage user's search for structures, trends, or testing a hypotheses through interaction. Students might also want to check their understanding of the material through self-assessment exercises. The system can periodically pose questions to the student. The simplest form is a tickler which is a question that pops up in random order but always in the appropriate context. Tickler questions focus the student's attention on specific issues and promote self-explanation as a means to improve comprehension. With tickler questions, neither the answers to the questions nor any sort of feedback is provided. Other questions that require student input can be placed at articulation points beyond which the learner cannot proceed until the question is answered correctly. Another technique is to specify a desired result and to have the users find ways of tinkering with the parameters of an algorithm to achieve the required result.

If on the other side, the algorithm animation system will be used to confirm or refute a hypothesis, or explain a concept, users' interaction with the system might be more stable and predictable and some of the parameters can be predetermined.

Which interaction mechanism is chosen for communication with the system depends on the users, their needs and tasks, but they should be as simple as possible so as not to overwhelm the user. For example, it is important to provide a graph editor/drawing utility if a novice will be specifying input data for a graph algorithm.

Interactive algorithm visualization systems should be forgiving to the user. In a highly-interactive system, there is always a situation when the user will press the wrong button, input invalid data, or manipulate the wrong graphic object. The vast majority of user errors occur because the developer of a system allows the error to occur. Most error messages therefore, can be eliminated by reducing the possibility of errors, by making sure that number fields only accept numbers, by providing lists wherever possible, by providing file selection dialogs rather than asking users to type filenames, and by providing default values. Default values let the user know the expected form of the input and consequently will speed up the input process. Prevention of errors requires that the designers anticipate the potential mistakes the user is likely to make. This is often the most difficult aspect of designing an effective user interface, since the designer's familiarity and knowledge of the program interferes with his or her ability to view the program as a novice might. Another way of dealing with erroneous input is to allow for easy reversal of actions. The *Undo* or *Redo* utilities relieve users' anxiety and encourage free exploration.

The interactability of the system is dependent on its response time and display rate. Time factor is important, because, on one hand, long response times (15 seconds and more) can result in user's frustration, annoyance or even anger. On the other hand, if the response is too quick (less than 1 second) it can result in the users learning less, reading with lower comprehension, and making more errors.

4 Concluding Remarks

This paper presented an overview of the important design issues and techniques for algorithm visualizations. The field of algorithm visualization is rapidly evolving. There are many systems available over the Internet and many are still under development. This paper is an attempt to bring to the attention of the visualization designers the importance of the preliminary step: the analysis of the users, their needs, and tasks, and its implications. We conclude this work by giving some recommendations derived from our experience in designing and using algorithm visualizations.

- Don't attempt to provide everything possible in the beginning. Provide what you can that is of real benefit to the user and is of high quality. Plan to incrementally add new features over time rather than implement everything at once.

- Keep in mind when designing new algorithm visualization that users are not interested in pretty pictures. They need something that will lead them to construct an empirical model of behavior. They should be able to relate the display of information to a context and connect the display to the environment from which it is derived.

- Make sure to prevent the animation from becoming too busy and too distracting, either spatially (too much is going on in parallel) or temporal (too much is changing too quickly). First, perform information analysis and then design the layout.

- Don't get carried away. Today's computer software provide many gadgets for designers to experiment with. It is so easy to pull down menus and select different typefaces, and assign vivid colors. People tend to use too many colors or simply choose the wrong color that degrades the presentation. A graphic overloaded design is confusing to users. Good graphic design will convey the intended message without distractions of any kind.

- To minimize students' startup learning time, provide standard GUI to manipulate the tools. Menus should appear in the same place, interactions should be consistent, and screen layouts should be similar. After several demonstrations of different tools, the students will be familiar with the interface and screen layout, and can concentrate on what is changing in the visualization.

- Provide tailor-made help files, directed to your users.

References

Baecker, R. (1998). Sorting out sorting: A case study of software visualization for teaching computer science. *Software Visualization: Programming as a Multimedia Experience*, 369–382.

Biermann, H., & Cole, R. (1999). Comic strips for algorithm visualization [NYU Technical Report 1999-778].

Brown, M. (1988). *Algorithm animation.* Cambridge, MA: MIT Press.

Brown, M., & Hershberger, J. (1992). Color and sound in algorithm animation. *Computer*, *25*(12), 52–63.

Brown, M., Najork, M., & Raisamo, R. (1997). A java-based implementation of collaborative active textbooks. *1997 IEEE Symposium on Visual Languages*, 372–379.

Cox, K., & Roman, G. (1992). *Abstraction in algorithm animation* (WUCS-92-14 Report). School of Engineering and Applied Science, Washington University in St. Louis.

Hundhausen, C. (1999). *Toward effective algorithm visualization artifacts: Designing for course.* Doctoral dissertation, University of Oregon.

Jackson, J., & Francioni, J. (1992). Aural signatures of parallel programs. *Proceedings of the 25th Hawaii Conference on System Sciences*, 218–229.

Khuri, S., & Hsu, H. (1999). Visualizing the cpu scheduler and page replacement algorithms. *Proceedings of the SIGCSE'99*, 227–231.

Khuri, S., & Hsu, H. (2000). Interactive packages for learning image compression algorithms. *Proceedings of the 5th ITiCSE*, 73–76.

Naps, T. (1990). Algorithm visualization in computer science laboratories. *Proceedings of the SIGCSE'90*, 105–110.

Simpson, R., Spalter, A., & Dam, A. (1999). Exploratories: An educational strategy for the 21st century. *Proceedings of the conference on SIGGRAPH 99: conference abstracts and applications*, 43–45.

Stasko, J. (1997). Using student-built algorithm animations as learning aids. *Proceedings of the SIGCSE'97*, 25–29.

Teller, S., Boyd, N., Porter, B., & Tornow, N. (1998). Distributed development and teaching of algorithmic concepts. *Proceedings of SIGGRAPH'98*, 94–101.