A User-Centred Approach for Designing Algorithm Visualizations

Sami Khuri

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 depictions of algorithms in execution are being used in explaining, designing, analysing algorithms, and in debugging, fine-tuning, and documenting programs. Although many tools have been developed over the past twenty years, little attention has been paid to the analysis of users, their needs, tasks, and goals. This paper gives a brief overview of the preliminary design stage of algorithm visualizations, namely the analysis of requirements.

Keywords: Algorithm Visualization, Educational Software, Visual Representation of Information, Design Guidelines, User Centred Approach

1 Introduction

Visualization is defined in the dictionary as "mentally visual images". In the field of computer science the term has a more specific meaning: "The technical speciality of visualization concerns itself with the display of behaviour, and particularly with making complex states of behaviour comprehensible to the human eye" [Gallagher 95]. The term was first made popular in the USA in a 1987 National Science Foundation initiative on scientific visualization that provided first definitions, goals and examples of visualization in scientific Computing [McCormick et al. 87]. Since then, numerous applications ranging from the visualization of mathematical and scientific data to the simulation of virtual environments have been developed. In computer science education, visualization tools can help instructors in a variety of ways, ranging from merely attracting students' attention to increasing students' understanding. Visualizations have been used in teaching, designing and analysing algorithms, producing technical drawings, debugging, fine-tuning, and documenting programs.

Many new algorithm visualizations are being developed each year, yet descriptions of visualization systems rarely specify any particular task they were intended to support [Petre et al. 98]. Creating educational algorithm visualizations requires substantial time and effort. Mapping an algorithm to an animated representation is a non-trivial problem; it requires careful thought and knowledge of a particular algorithm animation programming framework. Today's IDEs[1] provide many gadgets for designers to experiment with, and the emergence of the Java programming language allows them to make their systems available over the Internet. It is so easy to pull down menus and select different fonts, assign vivid colours and embed an applet in the Web page. Developers of educational visualization packages claim that their systems can be used by all kinds of users, novice and experienced, and in all kinds of tasks: demos, homework, laboratories, and self-study. But, no algorithm visualization will ever be universally superior across all kinds of users and tasks. In what follows, we describe a "user-centred" framework for designing algorithm visualizations, in which visualizations are developed for real needs, real users and real tasks, and not just because the technology to implement them is there.

1. IDE: Integrated Development Environment

2 Designing Algorithm Visualizations

Like many other design disciplines, a successful algorithm visualization design should consider the many facets of design issues such as effective representation and presentation, objectives, environmental considerations, design and layout, colour, graphics, and user interface. The process of producing algo-

Sami Khuri holds a M.Sc. degree in Mathematics, a M.Sc. in Computer Science and a PhD. in Computer Science, all three degrees from Syracuse University, USA. He is a Professor of Computer Science at San Jose State University, USA. He was the recipient of several research awards. He was a Dana scholar in the Biomedical Engineering Department at the Johns Hopkins School of Medicine, a Fulbright scholar at the American University in Bulgaria and a DAAD scholar at the Technical University of Munich in Germany. His research interests centre around the design and analysis of algorithms, genetic algorithms, visualization and algorithm animation. He published and presented his work at international Computer Science conferences and gave tutorials and lectures on Data Compression, Genetic Algorithms and Algorithm Visualizations. Sami Khuri has been involved in the design and implementation of interactive algorithm-specific visualizations used in Operating Systems, Compiler Design, Data Compression and Neural Network courses.

khuri@cs.sjsu.edu

http://www.mathcs.sjsu.edu/faculty/khuri/index.html

Part of this work was done during author's sabbatical leave at the Departamento de Lengajes y Ciencias de la Computación, Universidad de Málaga.

rithm visualizations is time-consuming and resource consuming and includes at least the following five steps: analysis of requirements, system design, implementation, testing, and maintenance. Obviously, these stages will often overlap and only a few systems pass smoothly through these steps. Testing, for example, may reveal ambiguities in the design specification, parts of which will have to be rewritten. Analysis of the requirements is always the most important stage, for it is the one that justifies the other four. Unfortunately, it is often the one that is skipped. For education software visualizations, this stage involves trying to provide questions, such as: Who will use the system? How would it fit into existing curricula? What would the system add to what could be done by other means? Is the system technically feasible? In what follows, we bring to the readers' attention the important parts of the analysis of requirements.

2.1 Users' analysis

Understanding who the users are determines system's content, organization, breadth, depth, and information presentation. Users of algorithm visualizations can be categorized into four roles: student, educator, researcher or developer (it is possible that one individual will assume more than one of these roles). Another possible approach is to characterize users into novice and experienced users. It is difficult to develop a system suitable for both novice and experts. The needs of programmers of different levels of expertise may not be satisfied by a single system. Users at the beginning levels do not learn well by trial and error (i.e. do not profit from "floundering" and trying to find their own way to correct paths after following incorrect ones for some time). They tend to have problems in mapping the real world model onto a program and they are easily distracted by motion and by extraneous detail. They easily form misconceptions and need as many of the following design features as possible:

- Graphical and consistent means of visualization control (e.g. menus, buttons, etc).
- Similar instructional structure of all visualizations, such as a number of worked examples showing how the algorithm can be used (build-in default visualizations), as well as a graphic support tool for working with new problems (e.g. the facility to input new data sets, change parameters of the algorithm, etc).
- Comprehensive help files with clear directions on how to use the tool, descriptions of the algorithm's steps, as well as the organization of the interfaces.
- Short "quizzes" or similar opportunities for students to evaluate whether they have understood the material, and to exercise the use of the visualization tools.

Expert users on the other hand, will want to "play" with the code, see how it works, and then modify, expand or otherwise just experiment with it. For example, they might want to see how they could integrate these modules with other tools. Thus, animations designed for experts might require the ability to move between algorithm-level and program-level displays depending on their needs.

2.2 Needs Analysis

After determining the group of visualization's users, it is very important to address their needs. To do so, it is necessary to realize the fact that different students have different learning styles and strategies. Several factors such as locus of control, motivation, and learners' expectation, all play a significant role in learning. Some students prefer the "hands on" approach, where they are actively involved in the learning process. These students tend to explore more and are generally more independent learners. However, some students prefer to be led through the lesson, allowing the instructor, or the computer to control the flow of the material. For the latter type of users, algorithm visualizations should graphically demonstrate effects of the algorithm on the data structures. It is also useful or even necessary, to support the animation with explanatory cues in form of short, on-screen, textual notes, and to provide control of the speed at which the algorithm is animated.

Support for users who prefer active interaction requires a tool-set similar to that found in program development environments. In particular, the availability of "debugging" style software to allow single step execution, breakpoint setting and the monitoring of key variables is essential. The ability to simultaneously view the algorithm execution path and the data structure state is a crucial aspect of interactive observation.

Before implementing an algorithm visualization system, the designer should also ask the following question: Do the users need this information presented in this way? For example, an instructor can illustrate the binary search algorithm by using a phone book. In general, effort is wasted on visualizing simple concepts or algorithms. For example, if the amount of data is small, or the data structure is very simple, or the relationship of objects is important but movement is not needed, then it is better to use a static picture. But for a large amount of data, and for complex data structures or when movement is needed to show how the relationships between objects change over time, animation is the right choice for the presentation of the algorithm.

2.3 Task Analysis

When designing an algorithm visualization system, it is important to note the system's intended goals and select the content accordingly. The user might use the system to create new animations, interact with existing visualizations to understand the behaviour of an algorithm, or visually debug programs. Each situation demands a different kind of visualization system, and it is difficult to build the system that satisfies all of them.

Traditionally, computer science instructors constructed visualizations that were later used either as visual aids in lectures, such as *BALSA* [Brown 88] or in closed laboratories, such as *GAIGS* [Naps 90]. More recently, computer science educators have advocated using algorithm visualization software, such as *XTango*, as the basis for visualization assignments, in which students construct their own visualizations of the algorithms under study [Stasko 97].

If a system is designed for classroom teaching it might intentionally show algorithm-level displays only and avoid pro-

gram-level displays in order to keep the student's minds off implementation details. Systems designed for user's exploration might require the ability to change settings, input different data sets, change speed, move one step back, or move between algorithm-level and program-level displays.

If novice users are expected to use the system to design their own visualizations, and if making animations is difficult and tedious, then their effort and time is wasted in low-level graphics programming. In one study, students spent over 33 hours on average constructing single visualizations with *JSamba* [Hundhausen 99]. The system designed for this purpose should have a powerful editor allowing students to map graphical objects to data structures automatically.

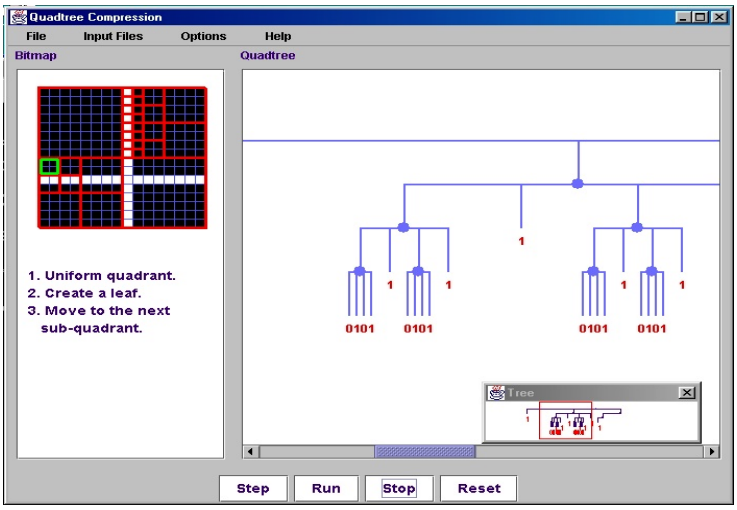

Fig. 1: Snapshot of the Quadtree package [Khuri/Hsu 00].

2.4 Information analysis

Information should be analysed to determine how effectively it could be visualized. Viewers should be able to forget about the technique of presentation and concentrate instead on what is being taught. Without an appropriate set of visual conventions, such as one colour to denote some items and another for other items, one may spend more energy trying to figure out what the picture means than in trying to follow the algorithm.

The graphical representation of information is heavily dependent on the concept we would like to visualize. For example, an important learning objective for students in understanding the bubble sort algorithm is to first comprehend the central metaphor of large values "bubbling up". Another learning objective is to understand the core operation of swapping two data values that sorting algorithms employ. 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 a graphical view of changing program data with a corresponding view of the executing source code. For example, the package shown in Figure 1 animates the quadtree compression algorithm for bitmap images. The quadtree method scans the bitmap, area by area, looking for areas filled with identical pixels. The package takes a bitmap as input, and constructs a quadtree, where a node is either a leaf or has exactly four children. The construction of the tree is done using a very interesting recursive algorithm. The area on the left-hand side of the application is used for displaying the bitmap image and steps of the algorithm. The red lines show how the quadtree algorithm partitions the image. A green rectangle indicates the quadrant being evaluated by the algorithm. The window on the right entitled "Quadtree" shows the building of the tree. A useful extension to the multiple windows approach is that of semantic zooming. The tool named "Tree" displays the whole quadtree but on a smaller scale than the one depicted in the "Quadtree" window. The current view of the "Quadtree" window is always the part of the tree inside the red rectangle in the display tool. Users can view other parts of the quadtree in the "Quadtree" window by simply moving the red rectangle to the desired location in the display tool.

In general, there are two approaches to visualizing different, but related algorithms. With the unified-view visualization, one graphical representation is used for different algorithms of the same class, and designers try to keep the individual adaptations as small as possible. The advantages of this approach is time saving since once the animation view has been established for the first algorithm, the views can then be reused. Because of the common base, the behaviour of related algorithms could be compared more easily. Another possible way of animating related algorithms is to use a unique view. In this approach, a unique representation is developed for each algorithm. It takes more time and effort to visualize algorithms this way because the best representation must be found for each algorithm, and the comparison of uniquely-visualized algorithms requires additional work, but the advantages of this method are meaningful graphical metaphors and increased memorability of the visualization. Care should be taken in selecting graphical metaphors. A given picture can mean several different things to different viewers, and the meaning will change depending on the relation to other simultaneously displayed images.

Developers have several choices of graphic representation [Cox/Roman 92]. In direct representation, data structures of an algorithm are directly mapped to a picture (e.g. representing an array as a collection of objects, where the index of each array element is mapped to the object's X-coordinate, while the value of the element is mapped to the object's Y-coordinate). In structural representations, some details or information are hidden and the remaining information is directly represented. For example, upon visualizing a computation running on a network of processes, we might want to reduce the complex states of the individual processes and simply show them as being in one of two states: "active" or "inactive". In this representation, we conceal the other attributes of the processes. Some representations can be of synthesized nature, i.e., the information of interest can be derived from the program data, but is not directly represented in the program. Some examples include counting the number of items already sorted or compression ratios. A

slightly different type of information representation is that of explanatory nature. In this type of representation, visual events have no counterparts in the underlying algorithm. They are added to enhance the presentation. Their goal is to communicate the implications of a particular computational event or to focus the viewer's attention.

Some of the many ways of representing information is by using shape, size, colour, texture, and arrangement of objects, sound, and 3D. Colour can be used to call attention to specific data, identify elements or structures, depict logical structure, increase the number of dimensions in coding the data, and highlight relationships. Although, colour can be a very powerful way of representing the information, some caution must be taken. With respect to colour, it seems best to be conservative. Only four distinct colours should be appropriate for novice viewers. This allows extra room in short-term memory (about 20 seconds), which can store up to five words or shapes, six letters, seven colours and eight digits. The same colour should be used for grouping related elements. It is important to be complete and consistent. For example, command and control colours in menus should not be used for information coding within a work area unless a specific connection is intended. Similar background colours of related areas can orient the viewer to understand the conceptual linking of the two areas.

Sound is another interesting area of research in information visualization. It is a useful complement to visual output because it can increase the amount of information communicated to the user or reduce the amount of information the user has to receive through the visual channel. Although not suitable for conveying exact values, auralization can indicate trends and increase the number of dimensions capable of being presented simultaneously.

2.5 Scope Analysis

The scope can range from single-purpose visualizations that illustrate one algorithm or a group of related algorithms in detail, to specialized systems that concentrate on algorithms in certain fields of computer science, such as graph algorithms and finally, to general purpose systems. The latter can (ideally) animate any algorithm. The greater the number of algorithms that can be animated, the more desirable the result. See [Khuri 00] for a collection of links to different algorithm visualization systems. Before embarking on developing a general-purpose system, one should consider that increased flexibility results in increased complexity. In general, systems that restrict themselves to animating only algorithms in one field, such as geometrical algorithms, might not be able to easily represent algorithms in other fields. On the other hand, specialized packages can be polished to make pleasing, informative visualizations of frequently used objects. Some of the recommendations for the designers are:

- Design small first. Don't attempt to provide everything possible in the beginning. Provide what you can that is beneficial to the user, visually attractive, and is of high quality.
- Plan a phased growth. The visualization might grow and change over time. Make sure to plan for growth by using object-oriented design and carefully documenting the programs. Some of the algorithm-specific visualizations might not allow adding new features. Plan to add new features over time and make upgrades publicly available.

2.6 Resource analysis

Remember that design takes longer than expected. Designing visualizations is not simple, especially when the designer is concerned with more than just the visualization appearance. Visualizations places great demands on computer resources, such as CPU speed, monitor size and resolution, RAM and disk memory sizes, networking, audio and video input and output, colour display panel and projection, and other input and output equipment needed. If visualization will be available over the Internet, they will need large amounts of storage space, and more importantly, a high-speed connection to the net.

Another purpose of the resource analysis is to select the specification method for designing new algorithm visualizations. Some of the possibilities are annotation, declaration, manipulation, and predefinition. In an annotation method, important steps of an algorithm are annotated with interesting events. The interested events call graphical operations which execute animations (move rectangles, change colours, etc.). When these program points are reached during execution, events are created and then forwarded to different views of the algorithm animation system. These views represent the interesting events by appropriate animations, as in *BALSA* [Brown 88]. In *PAVANE* [Cox/Roman 92], for example, the user can specify a mapping between the program's state and the final image arbitrarily or by declaration. The changes in the state will be immediately reflected in the image. This approach provides greater abstract capabilities, but this increased power also requires more processing to map the program to the final image. One of the interesting, but rarely found systems, is the one that allows the creation of new visualizations through manipulation (or animation by demonstration). The user can specify visualizations through the use of examples. The system attempts to capture the gestures used by the animator as she directly manipulates an image and ties these gestures to specific program events. This approach suffers from the difficulty of specifying the exact relationship between the gesture and the program event. The predefinition method is often used for application-specific visualizations, such as Quadtree in Figure 1, and employs a fixed or highly constrained mapping. There is little or no control over what is visualized and in the way the information is presented.

3 Concluding Remarks

In this paper, the preliminary design phase: the analysis of requirements is discussed. Many algorithm visualizations are being designed without paying too much attention to the needs of users, their tasks, and their characteristics. Creating visualizations requires substantial time and effort. Mapping an algorithm to an animated representation is a non-trivial problem; it requires careful thought and knowledge of a particular algorithm animation programming framework. There are many underutilized techniques for creating effective algorithm visualization, such as sound, 3D, and artificial intelligence, which

will continue to spark the creativity of the developers in the years to come. But the main problem still remains the same, there is no single algorithm visualization, specialized or general-purpose, that can satisfy all kinds of users, all kinds of tasks and can be used in all kinds of environments. Without the careful analysis of the users' needs, tasks, scope, information, and resources, the effort put in developing a visualization package will probably be wasted. Readers interested in obtaining more information about designing effective algorithm visualizations are referred to [Khuri 00] and [Stasko 98].

References

[Brown 88]
M. Brown: "Algorithm Animation", MIT Press, Cambridge, MA. 1988.

[Cox/Roman 92]
K. Cox and G. Roman: "Abstraction in Algorithm Animation", Technical Report WUCS-92-14, School of Engineering and Applied Science, Washington University in St. Louis, 1992.

[Gallagher 95]
R. Gallagher: "Computer Visualization: Graphics Techniques for Scientific and Engineering Analysis", CRC Press, 1995.

[Hundhausen 99]
C. D. Hundhausen: "Toward Effective Algorithm Visualization Artifacts: Designing for Participation and Communication in an Undergraduate Algorithms Course", Ph.D. Dissertation, University of Oregon, June 1999.

[Khuri/Hsu 00]
S. Khuri and H. Hsu: "Interactive Packages for Learning Image Compression Algorithms", Proceedings of the 5th ITiCSE, 2000, pp. 73–76.

[Khuri 00]
S. Khuri: "Designing Effective Algorithm Visualizations", Invited Lecture, Program Visualization Workshop, Porvoo, 2000, available at http://www.mathcs.sjsu.edu/faculty/khuri/invited.html

[McCormick et al. 87]
B. H. McCormick, T. A. DeFanti and M. Brown: "Visualization in Scientific Computing", Report of the NSF Advisory Panel on Graphics, Image Processing and Workstations, 1987.

[Naps 90]
T. L. Naps: "Algorithm Visualization in Computer Science Laboratories", Proceedings of the ACM SIGCSE, 1990.

[Petre et al. 98]
M. Petre, A. Blackwell and T. Green: "Cognitive Questions in Software Visualization", in Software Visualization, MIT Press, 1998, pp. 453–480.

[Stasko 97]
J. T. Stasko: "Using Student-Built Algorithm Animations as Learning Aids", Proceedings of the ACM SIGCSE, 1997.

[Stasko 98]
J. T. Stasko: editor, Software Visualization, MIT Press, 1998.