Why Johnny Won't Visualize

Eileen Kraemer
Computer Science Dept.
The University of Georgia
Athens, GA

Mihail Tudoreanu
Computer Science Dept.
Washington University
St. Louis, MO

Ashley Taylor
Computer Science Dept.
The University of Georgia
Athens, GA

Abstract

Many potential users of program visualization have a strong intuitive belief that visualization is a valuable tool for communicating information about the state and behavior of programs. Yet, in practice, the use of visualization is less pervasive than the notion that it is useful. Several reasons may exist for this discrepancy, among them the effort required to create and refine visualizations, difficulty in collecting information from running programs, difficulty of the viewers in navigating and refining the views they are presented with, as well as questions about the true benefits to the user. Our position is that interactivity is key to addressing many, though not all, of these concerns. In particular, we focus on supporting environments in which creators and users interact to create and refine visualizations in the same domain in which visualization occurs. In this paper we present three projects that take this approach and describe the planned empirical studies for evaluation of this work.

1 Introduction

While many programmers and programming students have a strong intuitive belief that visualization is valuable for communicating information about the state and behavior of programs, it seems that, in practice, program visualization is underutilized. One reason for the lack of use might be that creating visualizations is too difficult and time-consuming. Tools such as TANGO[2], POLKA[6], Pavane[7], Lens[8], VisAD[9] and others have attempted to remedy this problem. Still, the use of visualization seems not yet to be proportional to the general belief that it is useful. A number of factors may contribute to this apparent disconnect between belief in the usefulness of visualization and the extent to which visualization is actually employed.

In addition to creation, users and designers of visualizations must perform the tasks of navigation and refinement, often as a result of the insight gained from previous graphical representations. These otherwise natural tasks are often difficult and tedious, or may be limited by poor interaction mechanisms in the visualization tool. It is also common that, in order to modify the program view, the user must handle multiple tools such as graphics editors or compilers, each with its own cognitive domain, interaction style and assumptions about the user. Furthermore, the use of extra tools complicates the feedback loop defined by the place where an action is performed and the place where its result occurs. Often it is the case that a change in the visualization is performed outside the visualization environment, with the picture out of sight. The result of the change cannot be readily observed and adjustments become a complicated task.

A technical challenge to visualization involves linking the visualization system to the actual application. Further complicating the process is the use of off-the-shelf and third-party modules that either do not support monitoring or support other monitoring paradigms. A similar problem arises in the use of distributed systems that do not fall entirely under the administrative domain of the visualization user.

User's perceptions about visualizations often prevent users from employing them. One problem is the lack of trust in the stability of the visualization system, which might make users suspect that an application error is instead caused by the visualization. Another problem is the preconception/misconception among potential users that they must understand the program before beginning to visualize it. Finally, users may have doubts about the relative benefits of visualization given the perceived effort.

In this paper we present three projects addressing the concerns of difficulty in creation, navigation, and refinement and promoting both increased interactivity and the location of those actions within the domain of the visualization system. We then briefly describe an evaluation plan to address the value of these techniques.

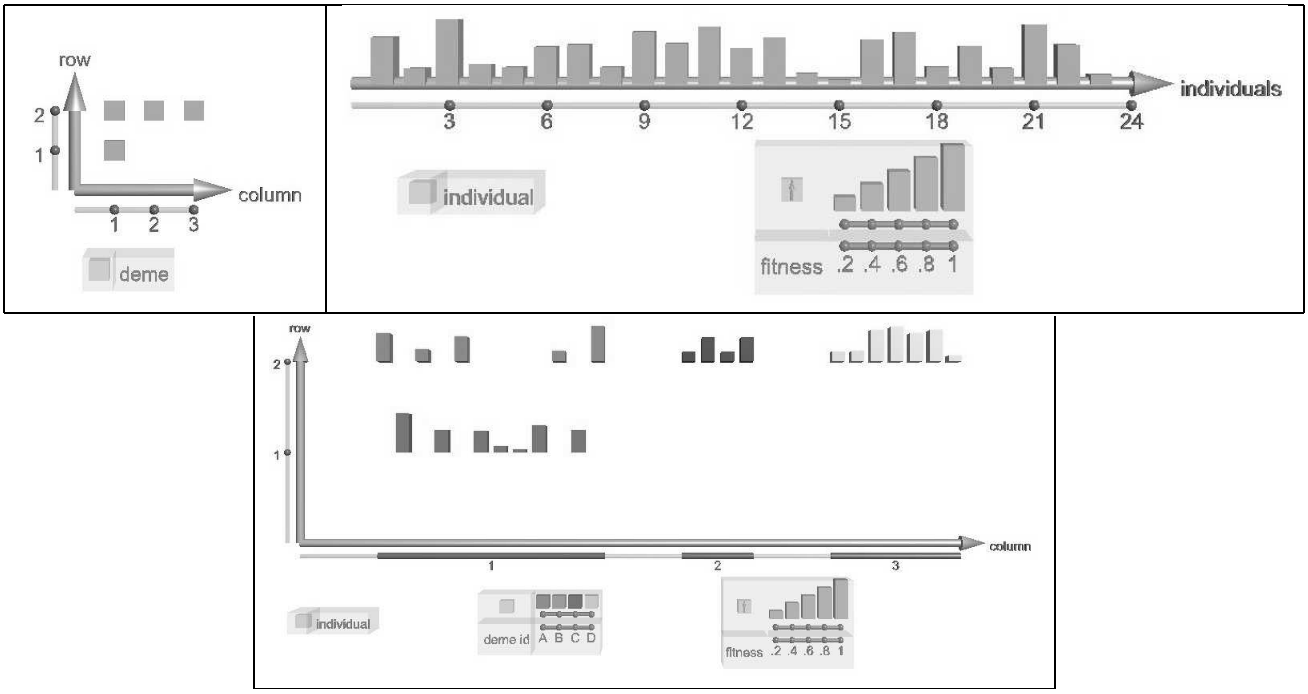

Figure 1: The upper left picture presents the row-column location of the demes (populations of individuals in a distributed genetic algorithm). The upper right figure shows the fitness of individuals of all demes. The picture at the bottom, that presents the individuals at their location, was obtained by first combining the other two visualizations.

2 Reshapeable Visualization

In this project we focus on providing a tool that addresses the problems related to creation and refinement of software visualizations. The tool tries to alleviate the need for additional programming and graphics tools, the often complicated and awkward manner in which visualizations are modified, the lack of power of the built-in interactions, and even the distinction between designer and viewer. Users can monitor and analyze the behavior of a running program while immersed in an environment that allows building and moving among visual representations. The representations can be customized to the monitored computation and to the user's momentary interest and understanding. Three features are detailed in the next paragraphs. First, the environment supports direct, on-screen manipulation of visualization, in most cases without the need for additional tools. Abstractions of the computation, i.e. animations that present information derived from, but not necessarily present in the computation can be built via a wide range of operations, including novel interactions with legends similar to those on a road map. Second, interactions are aimed at enhancing and simplifying user's control over the behavior of the visualization. Third, the environment also contains a mechanism for relieving the user of tedious tasks through the automatic generation of graphical design for displaying running programs.

In this environment, the user may often change the computation view to reflect the current user model and understanding of the computation. This change is performed mainly through direct manipulation of the items, i.e., animation components and program objects, that are pertinent to the intended change. The result of the action can be readily observed, which makes this interaction style very natural. It is possible that the natural feeling is given by a short feedback loop, and from the spatial proximity of an action to its result. The natural feeling might also come from maintaining the user focus in a single cognitive domain, as opposed to having to rely on multiple domains associated with multiple tools such as graphic or program editors, and activities such as compilation or error checking. Our supposition is that both a long feedback loop and changing cognitive domains impose a burden on the user and make a tool harder to use. Further research must be conducted to determine more precisely the impact of the two factors on the usability of a tool.

The operations in the environment manipulate either the underlying data and relations conveyed in the picture or the graphics that encode the information [5]. The visual abstraction of the program, the data and relations to be depicted, are derived from the program state based on the user actions, and may depend on multiple program states. When performing an operation, the user can consciously determine how the

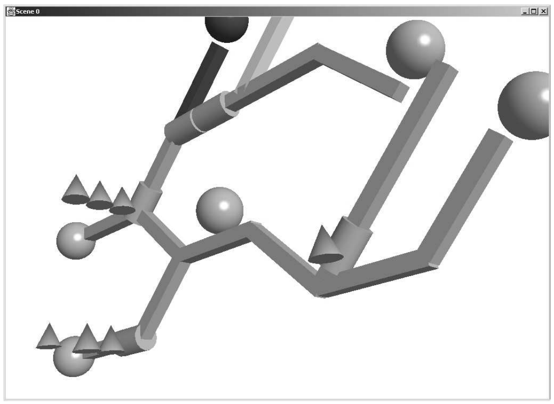

Figure 2: An automatically generated view of a railroad simulation. The cones at the same x and y location, that represent crates carried by one car (cylinder), are assigned different z-coordinates to avoid overlap.

visualization of future program states will be constructed with respect to the current and upcoming states. In effect, the future behavior of the animation relative to changes in the program state is defined. Another advantage of manipulating information is the query-like operations that help in identifying the items to be used by or modified via an operation. The user can specify directly the item (extensional query) or describe properties of the conveyed information for the items of interest (intensional query).

New interaction elements are provided through the visualization legends. Legends, including coordinate axes, appear by default in some visualization systems (APT[10] Sage[3]) to explain the mapping between graphical elements and the encoded data. In most cases, legends cannot be manipulated directly by a user (in Microsoft Excel [1] they can be used for selection). In our approach [11], via simple interactions such as sliding and drag-and-drop, the user can adjust and induce changes to the manner in which graphical features encode information. This permits, for example, the user to emphasize the information of interest, while keeping the current context. Critical information can be dynamically assigned a larger range of graphical attributes than the rest in order to make relative differences and trends easier to observe. Legends can also be used to derive new visualizations from other views of the program. Distinctive graphical features as well as the desired information can be combined since they are all part of the legend (Figure 1). Finally, the manner in which objects are animated can be shown in a legend by indicating how attributes of animated objects change over time. As such, the user can control the path and timing of the animation.

3 Automatic presentation of programs

Techniques for automatically generating graphical design are packaged in the Reshapeable Visualizations environment. Such techniques [4] can relieve the user from handling graphical details like the assignment of graphical attributes to encode certain information, or the positioning of graphical elements in a manner that avoids overlap (Figure 2). For most users and tasks, this feature also eliminates the need of a graphic editor by allowing visualizations to be created based only on a description of the program entities and relations to be shown. If the user has a partial mental model of the graphical design, e.g., the state must be displayed by color, the system can complete the design around user's expressed preferences.

The automatic presentation tool, though inspired by presentation tools for static data (APT[10]), is designed specifically for the dynamic information streaming from a running program. Three presentation goals are satisfied by the tool. The first goal, traditionally named *expressiveness* in APT[10], requires that each frame of the animation display all the entities and relations specified by the user and nothing else. The second goal, *visual classification*, ensures that the visualization distinguishes between different kinds of program entities. For example, in a distributed computation messages and machines would have different graphical representations that allow a viewer to easily infer their type, even when properties common to both types are displayed. The third goal, *continuity*, requires for the duration of a visualization session that the animation frames show similar data using common graphical features and unrelated data using different features.

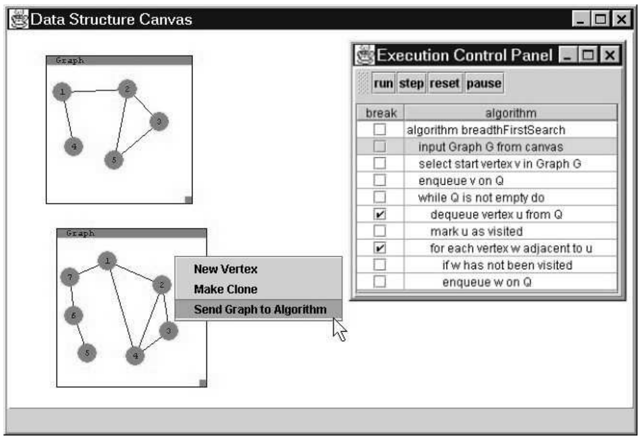

Figure 3: A user has constructed two graph instances on the SKA Canvas. The breadth-first algorithm on the right has been invoked and two breakpoints have been set. The algorithm is requesting an input graph, which the user is in the process of selecting.

4 SKA: Support Kit for Animation

Education is an important application of visualization. However, few studies of algorithm visualization (i.e. algorithm animation) use have shown positive results[12][13]. Most existing algorithm animation systems [2][6] were not designed to support interaction - they generally provide for the rendering of smooth animation, which is played like a video. We believe that the video paradigm does not engage students, and that this is one of the reasons for the poor animation study outcomes.

We have taken a fresh approach to the design of algorithm animation systems. We start by examining how instructors, tutors, and students discuss algorithms and data structures, and specifically how they use manual simulation. Our system, SKA (Support Kit for Animation), is intended to support and enhance the manual tasks performed during these discussions.

Tracing, or manual simulation, can be used for many purposes. It can demonstrate how an algorithm executes - a form of manual animation. It can be used to investigate how an algorithm handles special cases, and to explore variations of an algorithm. Tracing also allows a natural flexibility in the pace of execution.

However, tracing has limitations. It can be a long process, tedious and error-prone. Conceptual data structure diagrams are often used as input, and changes are recorded by erasing or overwriting the previous state. To record a visual history of changes requires that diagrams be redrawn repeatedly, and the time to do so tends to increase with the complexity of the algorithm and the diagram. The recording problem makes it hard for students to walk away with a detailed history of the trace to compare it to their reconstruction.

We also know that tracing takes place in a larger context. Part of this context is data structure diagram construction. Sometimes conceptual data structure instances are drawn just to show examples of the data structure, and are manipulated to show an operation (e.g. add an element) without explicit reference to an algorithm.

We believe that SKA can address the manual limitations we have noted, at least partially. Our initial design goal is to support common tasks such as drawing a conceptual data structure, manipulation of a data structure, execution of a high-level algorithm using a conceptual data structure diagram as input, and making copies of a data structure during execution to show the history of data structure states.

SKA is a combination of a semantic data structure drawing tool, an extensible library of data structure classes, and an animation action library. The drawing tool utilizes a canvas (Figure 3) on which users can construct and manipulate visual instances of predefined data structure classes. A visual data structure instance has a corresponding "real" data structure instance, and changing the contents of one causes an automatic change in the other. Users (instructors, tutors or students) can use the data structure instances as input to algorithms. Pseudocode algorithms can animate these data structure instances, but can also create data structure instances and display them on the canvas. Execution can be paused or single stepped, and breakpoints can be set and removed during execution, providing flexible user pace control. During pauses and at breakpoints, users can manipulate the data structure instances or clone them to record a history of states. The clones are independent instances, so they can also be manipulated, used as input, etc.

The authoring process is time-consuming with most existing algorithm animation systems[2][6]. SKA simplifies authoring by providing a library of common data structure classes, which handle interaction and layout automatically. The author changes the source program by replacing the data structure classes that they want to animate with the corresponding SKA data structure classes. At present, the program must be written in Java. The author writes a pseudocode version of the program in any pseudocode style. Calls are added to specify which points in the source program correspond to a particular pseudocode line. The program is then compiled with the SKA libraries. At runtime, the user sees the canvas and the pseudocode, which gives the feel of running a pseudocode algorithm directly on interactive data structure diagrams.

We are developing a framework to simplify the design of new SKA data structures. Finally, we are investigating the perceptual aspects of animation in order to inform the design for future versions of SKA and other systems, and to develop general authoring guidelines.

5 Evaluation of Usability Factors

Two properties of a software visualization tool need to be assessed: effectiveness and ease of use. The first property captures the degree to which the system can assist a user in gaining insight into a running program. Effectiveness can be characterized by both how correct the user's understanding of the computation is and how long it takes to achieve that understanding. The second property, ease of use, is linked to the burden a tool puts on a user. User's satisfaction with the tool provides a subjective measure for the overall ease of use. The degree to which the tool is intuitive can be evaluated by comparing the performance of users who received training for the tool with the performance of untrained users.

Our plan is to measure the differences in effectiveness and ease of use for program monitoring tools. The studies will assess how the existence of visualization, of interactive visualization, of a short feedback loop, and of multiple cognitive domains affects users. The main framework for these tests (except the feedback loop) consists of a number of tasks that will be performed by all subjects. Each task will involve a distributed computation because such computations are potentially harder to understand from the source code, which enhances the role of visualization. The number of tasks will be the same as the number of types of tools tested. For example, to test a system that only uses text-based monitoring versus a visualization system, two tasks are needed. Each task will be performed by some users with the help of a text only tool and by others with the support of the visualization. For interactive systems, we plan to study multiple interactive techniques in order both to obtain a more accurate estimate of the role of interaction in general and to compare the benefits of various interaction techniques.

To study characteristics of the feedback loop, we plan a more basic study that could be applied to a wider range of input methods, not only program visualization. The experiment will require users to perform a common tasks like writing or copying symbols. The equipment consists of electronic tablets and pens. The length of the feedback loop is varied by modifying the distance between the tip of the pen and the location where the "ink" is displayed. For various lengths, the accuracy of the user input will be evaluated.

6 Conclusions

Program visualization has the potential to simplify the task of program understanding. Barriers exist to the full utilization of this technology. We have developed tools that focus on increasing the quality of interactions with the visualization system and with localizing those interactions to the domain of the visualization component. The effectiveness of these tools is under investigation through a variety of user studies.

7 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 9619831 and 9996082. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Microsoft. Excel. http://microsoft.com/office/excel.

[2] John T. Stasko. TANGO: A framework and system for algorithm animation. *Computer*, 23(9):27–39, September 1990.

[3] Steven F. Roth and Joe Mattis. Automating the presentation of information. In *Proc. IEEE Conference on AI Applications*, pages 90–97, Miami Beach, FL, 1991.

[4] Mihail E. Tudoreanu and Eileen Kraemer. Automatic presentation of running programs. In *Proc. SPIE 2001 Conference on Visual Data Exploration and Analysis VIII*, San Jose, CA, 2001.

[5] Mihail E. Tudoreanu, Delbert Hart, and Gruia-Catalin Roman. Reshapeable visualizations. In *Proc. IEEE Multimedia Software Engineering*, pages 245–250, Taipei, Taiwan, 2000.

[6] John T. Stasko and Eileen Kraemer. A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing*, 18(2):258–264, June 1993.

[7] Gruia-Catalin Roman, Kenneth C. Cox, Donald Wilcox, and Jerome Y. Plun. Pavane: a system for declarative visualization of concurrent computations. *Journal of Visual Languages and Computing*, 3(2):161–193, 1992.

[8] Sougata Mukherjea and John T. Stasko. Toward visual debugging: Integrating algorithm animation capabilities within a source level debugger. *ACM Transactions on Computer-Human Interaction*, 1(3):215–244, September 1994.

[9] William Hibbard, Charles R. Dyer, and Brian Paul. Display of scientific data structures for algorithm visualization. In *Proc. IEEE Visualization*, pages 139–146, 1992.

[10] Jock D. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986.

[11] Mihail E. Tudoreanu and Eileen Kraemer. Legends as a device for interacting with visualizations. In Submission.

[12] Colleen Kehoe, John Stasko, and Ashley Taylor. Rethinking the evaluation of algorithm animations as learning aids: An observational study. *International Journal of Human-Computer Studies*, 54(2):265–284, 2001.

[13] M.D. Byrne, R. Catrambone, and J.T. Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers & Education*, 33(5):253–278, 1999.