Chapter 3.4
Applications of Visual Algorithm Simulation

Ari Korhonen
Helsinki University of Technology, Finland

ABSTRACT

Understanding data structures and algorithms is an integral part of software engineering and elementary computer science education. However, people usually have difficulty in understanding abstract concepts and processes such as procedural encoding of algorithms and data structures. One way to improve their understanding is to provide visualizations to make the abstract concepts more concrete. In this chapter, we represent a novel idea to promote the interaction between the user and the algorithm visualization system called visual algorithm simulation. As a proof of concept, we represent an application framework called Matrix that encapsulates the idea of visual algorithm simulation. The framework is applied by the TRAKLA2 learning environment in which algorithm simulation is employed to produce algorithm simulation exercises. Moreover, we discuss the benefits of such exercises and applications of visual algorithm simulation in general.

INTRODUCTION

Data structures and algorithms are important core issues in computer science education. They are also complex concepts, thus difficult to grasp by novice learners. Fortunately, algorithm animation, visual debugging and algorithm simulation are all suitable methods to aid the learning process. Much research has been carried out to identify the great number of issues that we must take into account while designing and creating effective visualizations and algorithm animation for teaching purposes (Baecker, 1998; Brown & Hershberger, 1992; Fleischer & Kucera, 2001; Gloor, 1998; Miller, 1993). See, for example, the techniques developed for using color and sound

DOI: 10.4018/978-1-61350-456-7.ch3.4

(Brown & Hershberger, 1992) or hand-made designs (Fleischer & Kucera, 2001) to enhance the algorithm animations. We argue, however, that these are only minor details (albeit important ones) in the learning process as a whole. In order to make a real difference here, we should change the point of view and look at the problem from the learner's perspective. How can we make sure the learner actually gets the picture? It is not what the learner sees but what he or she does. In addition, we argue that no matter what kind of visualizations the teacher has available, the tools cannot compete in their effectiveness with environments in which the learner must perform some actions in order to become convinced of his or her own understanding.

From the pedagogical point of view, for example, a plain tool for viewing the execution of an algorithm is not good enough (C. D. Hundhausen, Douglas, & Stasko, 2002; Naps et al., 2003). Even visual debugging cannot cope with the problem because it is always bound to the actual source code. It is still the system that does all the work and the learner only observes its behavior. At least we should ensure that a level of progress in learning has taken place. This requires an environment where we can give and obtain feedback on the student's performance.

Many ideas and systems have been introduced to enhance the interaction, assignments, mark-up facilities, and so on, including (Astrachan & Rodger, 1998; Brown & Raisamo, 1997; Grillmeyer, 1999; Hansen, Narayanan, & Schrimpsher, 2000; Mason & Woit, 1999; Reek, 1989; Stasko, 1997). On the other hand, the vast masses of students in basic computer science classes have led us into the situation in which giving individual guidance for a single student is impossible even with semi-automated systems. Thus, a kind of fully automatic instructor would be useful such as (Baker, Boilen, Goodrich, Tamassia, & Stibel, 1999; Benford, Burke, Foxley, Gutteridge, & Zin, 1993; Bridgeman, Goodrich, Kobourov, & Tamassia, 2000; English & Siviter, 2000; Higgins, Symeonidis, & Tsintsifas, 2002; Hyvönen & Malmi, 1993; Jackson & Usher, 1997; Reek, 1989; Saikkonen, Malmi, & Korhonen, 2001). However, the topics of data structures and algorithms are often introduced on more abstract level than those of basic programming courses. We are more interested in the logic and behavior of an algorithm than its implementation details. Therefore, systems that grade programming exercises are not suitable here. The problem is to find a suitable application framework for a system that is capable of interacting with the user through canonical data structure illustrations in this logical level and giving feedback on his or her performance. The aim is to extend the concept of direct manipulation (Stasko, 1991, 1998) to support not only manipulation of a visualization but also the real underlying data structures that the visualization reflects. It is a kind of combination of direct manipulation and visual debugging in which the user can debug the data structures through graphical user interface. Our approach, however, allows the user to manipulate the data structures in two different levels. First, in low level, the data structures and the data they contain can be altered, for example, by swapping keys in an array. Second, in higher level, the framework can provide ready-made algorithms that the user can execute during the manipulation process. Thus, instead of swapping the keys, the user can sort the whole array with one command. In addition, the high level algorithms can be simulated in terms of the low level operations. Thus, the simulation process can be verified by comparing it to the execution of an actual algorithm. Quite close to this idea comes PILOT (Bridgeman et al., 2000) in which the learner solves problems related to graph algorithms and receives graphical illustration of the correctness of the solution, along with a score and an explanation of the errors made. However, the current tool covers only graph algorithms, and especially the minimum spanning tree problem. Hence, there is no underlying general purpose application framework that can be extended to