Publication P2

Ville Karavirta, Ari Korhonen, Lauri Malmi, and Kimmo Stålnacke. 2004. MatrixPro – A tool for on-the-fly demonstration of data structures and algorithms. In: Ari Korhonen (editor). Proceedings of the Third Program Visualization Workshop (PVW 2004). University of Warwick, Coventry, UK. 1-2 July 2004. Coventry, UK. University of Warwick, Department of Computer Science. Research Report CS-RR-407. Pages 26-33. ISBN 0-902683-74-8.

MatrixPro – A Tool for On-The-Fly Demonstration of Data Structures and Algorithms

Ville Karavirta, Ari Korhonen, Lauri Malmi, and Kimmo Stålnacke
Helsinki University of Technology
Department of Computer Science and Engineering
Finland

{vkaravir, archie, lma, kstalnac}@cs.hut.fi

Abstract

In this paper, we introduce a new tool, MatrixPro, intended for illustrating algorithms in action. One can produce algorithm animations in terms of direct manipulation of the library data structures, the process we call visual algorithm simulation. The user does not need to code anything to build animations. Instead, he or she can graphically invoke ready-made operations available in the library data structures to simulate the working of real algorithms. Since the system understands the semantics of the operations, teachers can demonstrate the execution of algorithms on-the-fly with different input sets, or work with "what-if" questions students ask in lectures. Such an approach lowers considerably the step to adopt algorithm visualization for regular lecture practice.

1 Introduction

A wide range of visualization systems has been developed to demonstrate various computer science core topics in the past decade. The major problem, for example, in teaching data structures and algorithms has been the difficulty of capturing the dynamic nature of the material. A proper tool for classroom demonstration would provide an ideal way to teach these kinds of concepts. Such a tool would support custom input data sets (Brown, 1988), provide multiple views of the same data structure possibly with different levels of abstraction (Hansen et al., 2000; Stern et al., 1999), include execution history (Hung and Rodger, 2000), and allow flexible execution control (Boroni et al., 1996; Rößling and Freisleben, 2002; Stasko et al., 1993). However, even though there exist many very sophisticated tools to visualize and animate these topics, the systems do not support, in general, easy on-the-fly usage. The illustrative material is typically very laborious to create. Or, the material must be prepared beforehand – at least to some extent. Thus, very few systems support the demonstration *on-the-fly*.

In this paper, we describe how to apply *visual algorithm simulation* (Korhonen, 2003) to aid the development of illustrative material for a data structures and algorithms course. Visual algorithm simulation is a generalization of algorithm animation in the sense that it allows real interaction between the user and the underlying data structures. The user is not only able to watch an animation with different input sets but he or she can also change the same data structures the algorithm modifies during the animation process. Actually, the user is able to simulate the algorithm step by step if required. The simulation consists of the very same changes in the data structures as any implemented algorithm would do. Moreover, we extend the scope of direct manipulation (see, *e.g.*, Dance (Stasko, 1991) or Animal (Rößling, 2000)) in which the user can change the visualization on the screen: in visual algorithm simulation these changes are delivered to the underlying data structures that are updated as well. Thus, there is always an actual implementation for each data structure involved in the simulation process and the corresponding visualization appears automatically on the screen after the structure is changed by the algorithm or the user. In the rest of the paper, we use, for simplicity, the term algorithm simulation instead of visual algorithm simulation.

Previously, we have employed the concept of algorithm simulation in our TRAKLA2 system (Korhonen and Malmi, 2000; Korhonen et al., 2003) that delivers *algorithm simulation exercises*. The system can give immediate feedback on learners' performance by evaluating the

correctness of the algorithm simulation. The simulation concept allows the system to compare the user made simulation sequence with the sequence generated by an actual algorithm. In addition, the system can create a model solution as an animation for each individual exercise by executing the algorithm. Similar to this, we have developed an application that allows the instructor to use the same framework for illustrating several concepts in algorithmics. The new tool, called MatrixPro, is even more powerful because we do not have to settle for simulating the ready-made exercises and follow the strict algorithm involved, but we can also allow the instructor to interact with any data structure already implemented in our library. Moreover, the simulation may also follow an algorithm that has not been implemented yet.

MatrixPro is based on the Matrix *algorithm simulation application framework* (Korhonen and Malmi, 2002; Korhonen et al., 2002, 2001). The framework provides the basic visualization and animation functionalities that are employed in this application. MatrixPro allows an easy usage of these functionalities through the graphical user interface that is designed to support lecture use and easy demonstration requirements. The motivation is to build a general purpose tool that requires no prior preparation at all to illustrate several algorithms and data structures regularly taught in computer science classes.

The rest of the paper is organized as follows. We start by introducing some basic concepts of algorithm simulation in Section 2. In Section 3 we describe the MatrixPro tool that we have built to test our ideas in practice. Finally, Section 4 concludes the discussion and reports some preliminary evaluation results and our future development ideas.

2 Algorithm Simulation

In *algorithm simulation*, the user manipulates graphical objects according to the modifications allowed for the underlying structure (like an array or a binary tree) in question and creates a sequence of simulation steps. These steps include basic variable assignments, reference manipulation, and operation invocations such as insertions and deletions.

In *automatic algorithm animation*, the user may watch the display in which the changes in each data structure representation are based on the execution of a predefined algorithm. Thus, it is the algorithm that modifies the data structure in similar steps as above, and the visualizations representing the structure are generated automatically. In Matrix framework, both of these methods to create animation sequences are supported and the system allows them to be combined seamlessly.

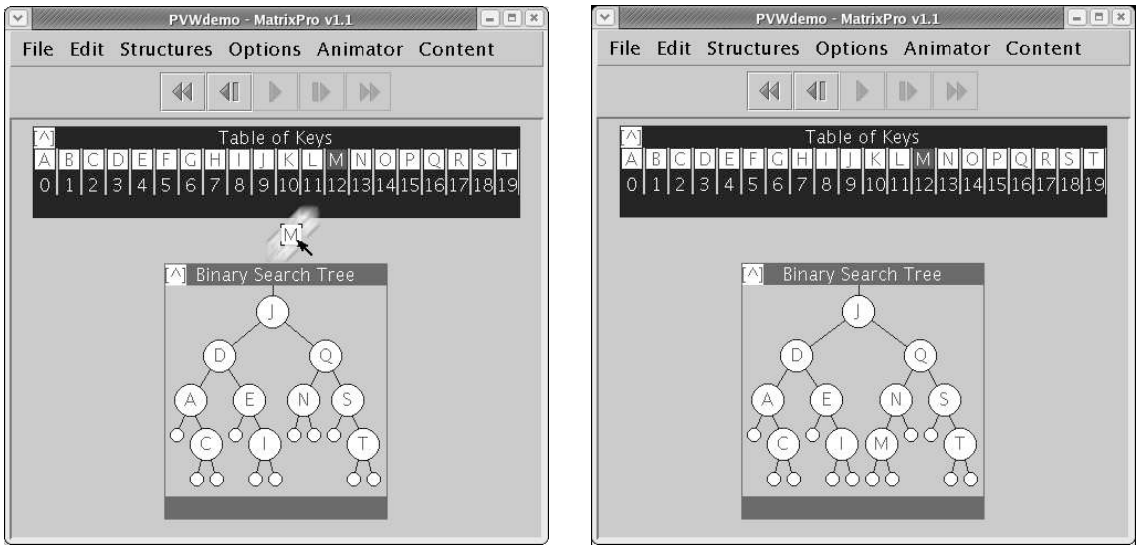
From the user point of view, algorithm simulation allows operations on a number of *visual concepts*. These include representations for arrays, linked lists, binary trees, common trees, and graphs. A data structure, such as a heap, can be visualized using several visual concepts like the binary tree or the array representation. Each visual concept may have several *layouts*, which control its visual appearance in detail. Moreover, the basic visual concepts can be nested to arbitrary complexity in order to generate more complex structures like adjacency lists or B-trees. The first one can be seen as a composite of an array and a number of lists, and the second one as a composite of a tree that has arrays in its nodes.

One of the key issues in algorithm simulation is that the user works on the *conceptual level* instead of the *code level*. The simulation environment automatically creates conceptual displays for data structures and allows the user to view and interact with the structures on different abstraction levels. Moreover, the actual underlying data structures are completely separated from their visual representations, and therefore one data structure can have different visual appearances. For example, a heap can be visualized either as an array or a binary tree, but the user can still invoke heap operations using both representations, even simultaneously.

In order to aid students to understand better the hierarchy of different concepts, we have introduced two separate groups of structures, Fundamental Data Types (FDT) and Conceptual Data Types (CDT). FDT structures behave like skeletons of data structures without any semantic information on the data stored in them (*e.g.*, lists, trees, and graphs). The struc-

ture is changed by performing primitive operations such as renaming keys or by reference manipulation. CDT structures are implementations for abstract data types, and they typically retain more constraints and are changed only by operation invocations that change the structure in terms of predefined rules. For example, Binary Search Tree (BST) is a structure that maintains the specific order of the keys stored in it.

An example of an algorithm simulation operation is represented in Figure 1. The example shows a simple operation of inserting one key into a binary search tree. In the operation, the user drags the key M from the Table of Keys and drops it on the BST (see Figure 1a). The system automatically calls the insert–routine of the underlying implementation of the BST, inserts the key to its correct position, and updates the visualization (see Figure 1b). The result is a valid binary search tree and its representation on the screen.



(a) (b)

Figure 1: Algorithm simulation example. (a) The user drags the key M from the Table of Keys and drops it on the binary search tree. (b) The key is inserted into the binary search tree.

3 System

MatrixPro is a tool for instructors to create algorithm animations in terms of algorithm simulation. The animations can be prepared prior to the lecture or on-the-fly during the lecture in order to demonstrate different algorithms and data structures at hand. The tool allows the instructor to ask, for example, *what-if* type of questions in a lecture situation, and thus make the lecture more interactive. Moreover, there is an option to introduce exercises for students (Korhonen et al., 2003). The details of these, however, are left out of the scope of this paper.

3.1 User Interface Objects

The main view of the program consists of a menubar, toolbar, and the area of the visualizations as depicted in Figure 2.

The menubar and the toolbar share the main functionality incorporated into the GUI. The menubar has the traditional File, Edit, and Options menus together with the special purpose Animator menu. There are also two other menus, Structures and Content menus, that are used to construct and insert *library data structures* on the screen, and to solve *algorithm simulation exercises*, respectively. The toolbar is an essential user interface component which

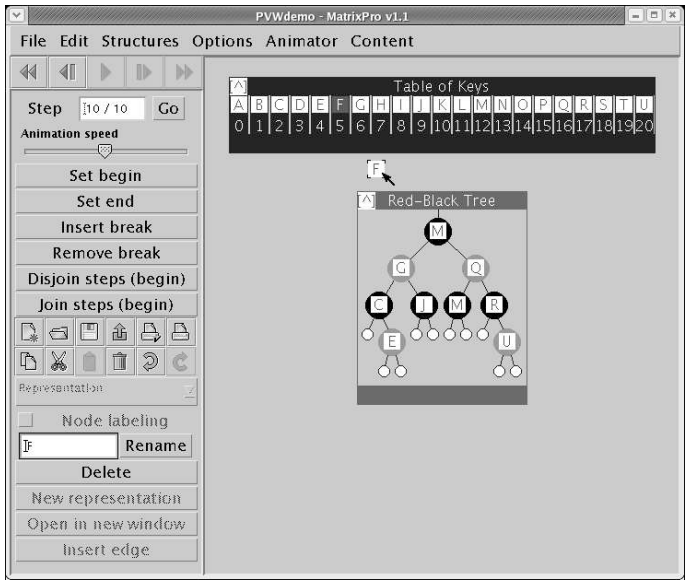


Figure 2: The MatrixPro main window. The user is currently dragging the key F to the Red-Black Tree.

enables users to handle the created animations. Through the toolbar the user can modify the animation easily by adjusting the animation speed, renaming representations, adding and removing breaks in an animation sequence, or changing the granularity of the animation sequence (join/disjoin steps). In addition, the toolbar contains functions for undoing and redoing operations as well as saving and opening animations.

Both – the toolbar and the menubar – contain the operational interface of the system called the *animator*. All built-in structures are implemented in a way that each update operation, such as a method invocation that changes the structure, is stored step by step in an additional queue and can be revoked later on. This technique allows the animator to rewind the queue of steps and replay the sequence. Naturally, the steps can also be invoked and revoked one at a time.

The area of visualizations contains the *visual entities* that the user can interact with in terms of algorithm simulation. Each visual entity is composed of at most four different types of interactive components. A *hierarchy* refers to a set of *nodes* connected with *binary relations*, and a node can hold a *key*. A key can be a primitive or some more complex structure. If the key is a primitive, it has no internal structure that could be examined further. If the key is some more complex structure, it is treated recursively as a hierarchy.

All these four entities can be moved around the display. The simulation consists of *drag and drop operations* which can be carried out by picking up the *source* entity and moving it onto the *target* entity. However, only the end point of a reference (binary relation) moves. Each single operation performs the *proper* action for the corresponding underlying data structure the entity is representing. An action is proper if the underlying data structure object accepts the change (*e.g.*, change of a key value in a node or change of a reference target).

A collection of hierarchies can be combined into a single window. Within a window it is possible to move entities from one hierarchy to another as described above. By using the GUI clipboard functionality, entities can also be moved between windows. In addition, the GUI provides functionality for manipulating hierarchies such as opening a (sub)hierarchy in other window, disposing of a hierarchy, minimizing or hiding an entity or some part of it, (re)naming or resizing an entity, changing the orientation of a hierarchy (flip, rotate), and changing the representation (concept or layout) for a hierarchy.

3.2 Features

On-the-fly usage. One of the most important features is the ability to use the system on-the-fly basis. MatrixPro offers an easy way to create algorithm animations by either applying the automatic animation of CDT structures or by simulating an algorithm by hand. All the ready-made CDT structures can be animated on-the-fly by invoking operations on them. Moreover, the user can freely modify the data structures also as an FDT by making primitive changes.

The system aids the algorithm simulation process by making it easy to invoke operations for structures also from the toolbar. By implementing a simple interface any method of the underlying structure can have a corresponding interface functionality (*e.g.*, push button) that appears both in the toolbar and in the pop-up menu of the structure. For example, for AVL trees the automatic balancing after insertion can be turned off and the rotations simulated by push buttons when appropriate during the simulation process.

In addition, the nodes in a structure can be automatically labelled, *i.e.*, a unique number appears beside each node. With this feature turned on, one can explicitly refer to a node while explaining or asking something. This is useful especially in a lecture situation as the instructor can ask questions concerning the view.

Customized animations. The system supports customization of animation in two ways. First, the user can build animations with custom input data sets by simply dragging elements from one structure, *e.g.* an array, to another structure. Even the whole array can be inserted into another structure, in which case the items are inserted in consecutive order to the target, if it supports the insertion operation. Second, the system allows controlling the granularity of the visualized execution history, *i.e.*, how large steps are shown when browsing the animation sequence. For example, primitive FDT modifications usually correspond to a couple of micro steps that form a single animator step. CDT operations, however, may consist of many animator steps. Thus, a number of animator steps can be combined to a macro step that is performed with a single GUI operation. In general, one animator step can have any number of nested steps.

Storing and Retrieving Animations. Although the automatically created animations may serve as visualizations to illustrate an algorithm without any modifications, some instructors certainly want to customize and save examples for later use. The *underlying data structures* can be saved and loaded as serialized Java objects. Thus, the corresponding animations can be recreated from this file format. Moreover, the *animations* can also be exported in Scalable Vector Graphics (SVG) format. The generated SVG animations also include a control panel that enables to move the animation backward and forward in a similar way as in MatrixPro itself.

In some cases teachers need to prepare simple figures of data structure, *e.g.*, to illustrate textual material. Of course, general purpose drawing tools and formats can be applied, but they lack the ability to automate the process of creating data structure representations by directly executing the corresponding algorithms that produce the result. For example, the TEXdrawmacros can be considered to be such a format that the user or a tool can produce. However, creating a complex conceptual visualization (for example, a red-black tree with dozens of nodes and edges or even a directed graph) is a very time consuming process without a tool that can be automated (*i.e.*, programmed) to produce valid displays. With MatrixPro this process can be easily automated and the produced visualizations can be exported directly in TEXdraw format to be included in LATEXdocuments.

Finally, we note that data structures can also be loaded from and saved into ASCII files which is practical, for example, for graphs, because the adjacency list representation is easy to write using any text editor. However, in this case, only the FDT information is stored.

Figure 3 summarizes the working process when preparing examples.

Customizable user-interface. The user interface can be customized to fit the needs of

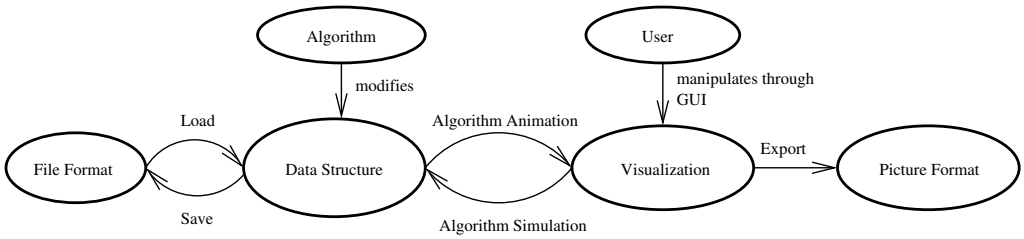


Figure 3: The process of creating algorithm animations in terms of algorithm simulation.

various users as the initial set of toolbar objects can easily be modified.

In addition, by modifying the configuration file, the contents of the menubar and pop-up menu can be changed. For example, the default font, font size, and background color can be set. Moreover, the default representation (layout) for each data structure can be changed.

Library. MatrixPro includes a library of data structures, which the user can operate on.

4 Conclusions

Algorithm animation has been an active research topic since the early 1980's and a multitude of visualization systems has been developed during this time. However, there has been no breakthrough in the field so that visualization systems would have become standard tools for instructors. One of the key reasons for this has been that creating animations is too laborious. Simply, installing the systems, learning to use them, and finally preparing examples take much more time than, for example, simple drawing on a whiteboard.

In this paper, we have introduced a system, MatrixPro, which overcomes the preparation barrier by applying algorithm simulation and a simple user interface. The power of algorithm simulation is based on the following issues. First, data structures are presented and manipulated on a conceptual level, which gives the teacher an opportunity to concentrate on the key concepts and principles of the algorithm. No coding is needed to create dynamic examples of algorithms. In addition, code level details which complicate student's learning process are totally suppressed. All this supports students' learning process and helps them to build viable mental models about the topics at hand. Moreover, the classification of structures to fundamental data types, conceptual data types and abstract data types helps them to better understand the hierarchies among different structures. Actually, most text books do not support this since they classify algorithms solely on the application point of view, *i.e.*, to basic structures, sorting, searching and graph algorithms.

Second, algorithm simulation is a far more powerful interaction method than that allowed by simple drawing tools. In algorithm simulation, the system interprets the simulation operations in the appropriate context. One can perform either simple low level operations such as assigning a new key value to a node or changing a reference, or higher level operations on abstract data types. Because no coding is required, it is straightforward to demonstrate on-the-fly what happens for a structure when operations are performed on it. This allows easy exploration of the general behavior of the algorithm with different sets of input values. Moreover, the teacher (or a student) can easily ask what-if type questions and get an immediate response from the system. Finally, algorithm simulation allows creation of exercises with automatic feedback. We have prepared an extensive set of such exercises in a system called TRAKLA2 (Korhonen et al., 2003) and these exercises are also available in MatrixPro.

Because generation of examples remains a natural task for teachers, MatrixPro supports tuning the presentation output and dynamics in many ways. The resulting ready animations or figures can be stored in formats which apply well to publishing them on papers or web pages. This is an important practical point of view, and neglecting it would diminish the

advantages of using any visualization system considerably.

Our own experience from the system has been very positive. For example, demonstrating complicated dictionaries such as red-black trees or B-trees, has previously been very clumsy using drawing tools. With MatrixPro it is easy to show the basic principles and describe the general behavior of a data structure in a very obvious way. Based on these experiences we believe the approach demonstrated in the MatrixPro tool is the direction that finally breaks the barrier of using algorithm visualization tools widely.

However, there are still many possibilities for improvement. First, although we already cover the most important dictionaries, we need to widen the current selection of priority queues, sorting methods and graph algorithms as well as other application fields. Second, some sort of history view showing several steps simultaneously is needed to make it easier to visually compare the states of the structure. Third, in many cases, especially with graph algorithms, it is important to allow free positioning mode for the user to create visually appealing examples.

Finally, we note that an obvious barrier remains: it is hard or impossible to transfer pictures or animations from one visualization system to another. People prefer to use different systems and these should not stay isolated from each other. Thus, we should be able to continue processing of an existing animation in a system that better supports the features we require. However, this is an issue that should be discussed on a wider forum among the developers of the visualization systems.

References

Christopher M. Boroni, Torlief J. Eneboe, Frances W. Goosey, Jason A. Ross, and Rockford J. Ross. Dancing with Dynalab. In *27th SIGCSE Technical Symposium on Computer Science Education*, pages 135–139. ACM, 1996.

Marc H. Brown. *Algorithm Animation*. MIT Press, Cambridge, Massachussets, 1988.

Steven R. Hansen, N. Hari Narayanan, and Dan Schrimpsher. Helping learners visualize and comprehend algorithms. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, 2(1), May 2000.

Ted Hung and Susan H. Rodger. Increasing visualization and interaction in the automata theory course. In *The proceedings of the 31st ACM SIGCSE Technical Symposium on Computer Science Education*, pages 6–10, Austin, Texas, USA, 2000. ACM.

Ari Korhonen. *Visual Algorithm Simulation*. Doctoral thesis, Helsinki University of Technology, 2003.

Ari Korhonen and Lauri Malmi. Algorithm simulation with automatic assessment. In *Proceedings of The 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pages 160–163, Helsinki, Finland, 2000. ACM.

Ari Korhonen and Lauri Malmi. Matrix — Concept animation and algorithm simulation system. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 109–114, Trento, Italy, May 2002. ACM.

Ari Korhonen, Lauri Malmi, Jussi Nikander, and Panu Silvasti. Algorithm simulation – a novel way to specify algorithm animations. In Mordechai Ben-Ari, editor, *Proceedings of the Second Program Visualization Workshop*, pages 28–36, HorstrupCentret, Denmark, June 2002.

Ari Korhonen, Lauri Malmi, and Panu Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In *Proceedings of Kolin Kolistelut / Koli*

Calling – Third Annual Baltic Conference on Computer Science Education, pages 48–56, Joensuu, Finland, 2003.

Ari Korhonen, Jussi Nikander, Riku Saikkonen, and Petri Tenhunen. Matrix – algorithm simulation and animation tool. http://www.cs.hut.fi/Research/Matrix/, Nov 2001.

Guido Rößling. The ANIMAL algorithm animation tool. In *Proceedings of the 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'00*, pages 37–40, Helsinki, Finland, 2000. ACM.

Guido Rößling and Bernd Freisleben. ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages and Computing*, 13(3):341–354, 2002.

John Stasko, Albert Badre, and Clayton Lewis. Do algorithm animations assist learning? An empirical study and analysis. In *Proceedings of the INTERCHI'93 Conference on Human Factors on Computing Systems*, pages 61–66, Amsterdam, Netherlands, 1993. ACM.

John T. Stasko. Using direct manipulation to build algorithm animations by demonstration. In *Proceedings of Conference on Human Factors and Computing Systems*, pages 307–314, New Orleans, Louisiana, USA, 1991. ACM, New York.

Linda Stern, Harald Søndergaard, and Lee Naish. A strategy for managing content complexity in algorithm animation. In *Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education, ITiCSE'99*, pages 127–130, Kracow, Poland, 1999. ACM Press. ISBN 1-58113-087-2.