T U M

I N S T I T U T F Ü R I N F O R M A T I K

How Helpful Are Systems for Algorithm Visualization?

Michal Mnuk

TUM-I0303
April 03

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

How Helpful Are Systems for Algorithm Visualization?

Michal Mnuk
Institut für Informatik
Technische Universität München
D-80290 München, Germany
mnuk@in.tum.de
http://wwwmayr.in.tum.de/~mnuk

April 3, 2003

Abstract

Many undergraduate courses in computer science can be endorsed by visualizations of algorithms. However, current systems provide only a limited support for efficient creation of high quality visualizations. We analyze the requirements on such a system from different points of view and propose guidelines that would foster an easy creation of didactically and graphically sound visualizations.

1 Goals

The modern style of teaching basic level undergraduate computer science courses should dramatically differ from that one practiced several years ago. Today's lecture halls are equipped with computers, graphical displays, beamers, wireless networks and other facilities that are predestined to open the door for multimedia in teaching in all variations. The times when blackboard and chalk were the only tools used is supposed to be definitely over. But they are not. Not yet.

In this paper we focus on basic undergraduate courses in computer science – those where basic data structures, searching, sorting or graph algorithms are taught. We discuss the question how to utilize animation techniques to teach algorithms more efficiently in this setting. We analyze available tools and propose a new architecture that allows us to make visualizations of high didactical and graphical quality and that does not exhibit the deficiencies found in currently available systems.

Before we proceed, we specify the setting and conditions that this paper is based on for they decisively influence the form of presentation of visualizations.

2 Starting Conditions

Basic undergraduate courses have a number of characteristic features:

- The students are beginners in early stages of their education. Many of them are faced the notion of an algorithm for the first time.
- The algorithms that are taught are reasonably simple and the ideas behind them can easily be conveyed as well.
- The data structures that occur are simple and suitable to be represented by pictures.
- Students are required to understand the presented algorithms quickly, they must remember and be able to actively use them throughout their study and far beyond it.

Studies have shown (e.g. [Ben-Ari(2001)]) that at this level a proper presentation of algorithms in visual form may substantially foster the learning process. However, in too many cases the old-fashioned style of teaching is still prevailing – some code fragments are written on the blackboard and a couple of pictures are drawn to illustrate how they work.

Nevertheless, the majority of lecturers use pictures when explaining algorithms, probably due to their belief that presenting information in its visual form is useful or even necessary. That is why many, more or less elaborated, tools for visualization of algorithms appeared in the past. But when we analyze the requirements on a good visualization at the basic undergraduate level, it is hard to find a tool that would satisfy all needs. Currently available systems, even though they have a number of sophisticated features, still fail to provide an appropriate working environment.

3 Requirements on a Visualization System

Before we start to postulate requirements on a visualization system, we want to clarify a question that the reader will be asking at this point: “Why do we need yet another visualization system? Are there not enough of them already?”.

The orientation of visualization systems differ greatly depending on their purpose. We aim at a situation where students are presented unknown algorithms for the first time – as in any introductory course. The algorithms should be presented in an easy and understandable way without relying too much on coding in a particular language.

There are several other reasons that make this situation distinguished:

- Since there are usually many algorithms to be presented in an introductory course by a lecturer, the time to be spent on preparation of a visualization of a single algorithm is limited.

- Every lecturer has his/her own preferences how to present an algorithm. Hence, even if there are visualizations available, it will be necessary to make at least minor adjustments. The effort to accomplish this must be kept minimized.
- A non trivial algorithm is usually not presented in one shot. Good lecturers will first convey the principles without going too much into detail. Only after that they will explain implementation issues. Thus, a proper presentation comprises several abstraction levels.
- At the time of the presentation, the students have, in general, only a limited level of understanding of the algorithm. Therefore, the visualization must be to a large extent self explaining. This is also a prerequisite for a visualization to be distributed to other people and it makes a visualization suitable for self-study.

To sum up, a versatile easy-to-use system for making visualizations is required. Even though there are quite a few software packages available, there is none that would fit well enough. This is one of the reasons why good visualizations of algorithms are still rare in computer science courses.

4 What Is a Good Visualization System?

The question whether a visualization system is good or bad is meaningful only with respect to a particular purpose and setting. And even then, perception is inherently subjective and hence a universally valid answer to this question cannot exist. Nevertheless, there are some general guidelines how to design visualizations (see e.g. [Khuri(2000a), Ben-Ari(2001)]) that yield satisfactory results in many cases.

Recall that we focus on a situation where students are presented unknown algorithms. In this setting, a visualization should clearly convey all important principles and supply enough details that would enable students to write their own implementation. On the other hand, to respect also lecturer's interests, a visualization must be easy to create and to adapt to various needs.

Now we are going to analyze the question what is a good visualization system from two different points of view – that of a viewer and that of a lecturer. A viewer is primarily interested in obtaining a well structured information from which both the principle and the details of an algorithm could be extracted. A lecturer wants to have a toolkit that would be easy to work with and would take over as many tasks as possible. We require that a visualization system takes the interests of both parties into account.

4.1 Visualization from the Perspective of the Viewer

An algorithm is a sequence of steps that produce some output from supplied input data. To "learn" an algorithm means both to remember this particular sequence of steps and to understand the reasons why it works the intended way. Of course, a visualization can not replace the proof of correctness but it is not our intension to use visualizations for this purpose. Being presented an algorithm and its impact on the data, the students can develop a thorough understanding of it.

A well prepared visualization lives from the clarity of conveyed information. At this place we recall an important rule for communicating ideas in talks that applies not only to a scientific environment:

Tell them, what you are going to tell them.
Tell them.
Tell them, what you have told them.

Even though the whole information is contained in the "tell them" step and the other steps are just repetitions, it is generally agreed that the first and the last step make the understanding much easier.

A visualization is in principle nothing else than a talk. Only the media is different. Thus, the ingredients for good visualizations are very similar to those for good talks. The above rule adopted to visualizations could then read: Show them, what you are going to show them – Show them – Show them, what you have shown them.

Despite of many similarities, talks and visualizations are not the same. While an aural information is basically ubiquitous (perceivable e.g. in the whole lecture hall), a visual information is available only at a very restricted location (e.g. certain, possibly small, area of the blackboard or the screen). Thus, in order to achieve a similar positive impact, the above rule needs to be extended to:

Show them, what you are going to show them and where it will happen.
Show them.
Show them, what you have shown them and where it happened.

We strongly believe that iterating this rule is a basis and a prerequisite for any good visualization. Unfortunately, only a negligible percentage of visualizations that we studied were made this way.

Now, we consider the three steps in detail, elaborate on why they are important and show how to enforce them.

4.1.1 Step 1: Show them, what you are going to show them and where it will happen

The importance of this point is extraordinary. The less acquainted the viewers are with the underlying algorithm, the more important this step is. Viewers that are already familiar with the algorithm know where to focus their attention and what to expect. Paradoxically, this issue is almost completely neglected in current visualizations, and systems provide no support to implement it easily. This is the major reason why it is difficult to follow many visualizations.

Before something important happens on the screen, it is crucial to attract the attention of the viewer to that area. In general, several means are at our disposal – textual, graphical, aural, etc. We propose to achieve this goal using a combination of two of them.

First, every visualization should be accompanied by a more or less abstract description of the algorithm (we will discuss this point later). At each step, the viewer is presented the part of the algorithm that is currently executed. This supplies a textual hint about what is going to happen next. Second, having this information, the viewer should then be visually led to the area where some changes occur.

Let us suppose that we are going to visualize some sorting algorithm that uses swaps to bring the elements of an array into the correct order. Figure 1

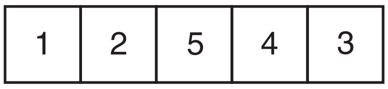

Figure 1:

shows an array where the third and the fifth element are supposed change their positions.

Now, we apply the reasoning from above to this example. A high level description of the algorithm available on the screen, such as in Figure 2, tells the viewer what to expect.

```
...
swap elements $A_i$ and $A_j$;
...
```

Figure 2:

When the values of i and j are known, the above code fragment states clearly which elements will be exchanged. This information helps to interpret the visual action that will follow.

For viewers who were not able to keep track of the values of i and j and/or to repeat the message by visual means, the third and the fifth element will be "highlighted" (e.g. by drawing them bold or in different color) and all other elements will be "brought to the background" (e.g. by decreasing the saturation or drawing them in a light tint (see Figure 3).

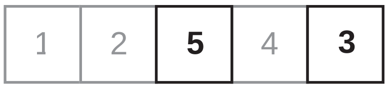

Figure 3:

The combination of a visual and a textual information makes sure that a visualization is unambiguous thus giving the viewers a chance to actively learn from what they see.

Let us consider another example. Graph algorithms often require to traverse the neighborhood of a vertex. In that case, it would not make much sense to tell the viewer each time to look at the vertex 7, vertex 11, etc. Instead, it is enough (and much better) to use statements such as

```
**for** all vertices in the neighborhood of vertex $v$ **do**
  ...
**end**
```

Figure 4:

After seeing this, the viewer knows that the algorithm will go through the neighborhood of v, and if we stick to the rule that neighbors are selected in the clockwise order, the viewer has a complete information to anticipate where the main focus will move.

Nevertheless, it is important to repeat this information also on the visual level. The processing could then be visualized by "smoothly" selecting vertices in the neighborhood one by one.

Thus, when watching visualizations, at the beginning of major steps the viewer should always be given a hint what to expect and where things are going to happen. This knowledge makes it not only much easier to comprehend what is happening, but also the viewer can check whether the observed behavior correspond to own expectations. In this way a properly prepared visualization facilitates understanding and, at the same time, serves as a check.

4.1.2 Step 2: Show them

In this, undoubtedly most important, part of any visualization the viewer is presented how the algorithm works. Unfortunately, the majority of available systems reduce the whole process of visualization to just this step.

Numerous papers dealing with the problem how to design visualizations provide a broad spectrum of possibilities how to "show them" things in a proper way ([Naps(2001), Khuri(2000a), Khuri(2001)]). There are several generally valid guidelines that should be followed in almost all cases. Here are some of them:

- Keep the information simple and consistent. Present it in multiple ways.
- Define presentation rules and stick to them.
- Do not overload the display.

- Utilize media carefully.

But no matter in what environment the presentation is done, it must always be kept in mind that the primary goal is to present an algorithm, not just nice pictures.

We add some more items that are usually not addressed at all or not dealt with in a proper way in available visualization systems.

Using Motion. The fundamental assumption in visualization is that the sequence of steps that an algorithm performs can easily be deduced by observing changes on the data. We use the changes as the main mean to communicate the way how an algorithm works. Hence, it is crucial to make them as clear and as explicit as possible.

Even though every algorithm is inherently dynamic, many visualizations present only a sequence of pictures representing the "states" of the algorithm. This is not only inappropriate. Without additional information it makes it cumbersome for the viewer to deduce what action may have caused a particular change on the data (the situation can be partially relieved by obeying the rules discussed in Section 4.1.1).

Figure 5 illustrates this approach. The picture on the left shows the data before the change, that on the right the data after the change. Since both pictures look on the first sight the same, they do not immediately evoke a feeling that something changed. The viewer is then forced to carefully analyze these picture in order to be able to find the difference. Usually, swapping the values 3 and 5 would be visualized in this (obviously wrong) way.

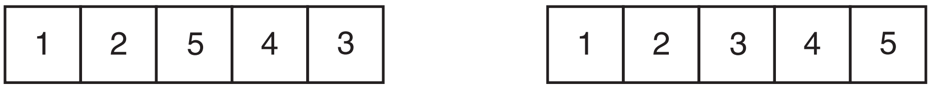

Figure 5:

In reality, the situation is even worse. The viewer sees only one picture/state at a time. Hence, the prerequisite to be able to tell the difference at all is that he or she remembers at least the last picture. This is not acceptable. Visualization should facilitate the process of learning and understanding algorithms. It must not become an additional obstacle.

In [Biermann and Cole(1998)] a visualization system is presented that addresses this issue. It shows two (or more) pictures/states at a time to make it easier to recognize the differences. But, in fact, it is only an attempt to make a wrong approach more useful.

The fact that the most steps in an algorithm are dynamic in their nature is a clear suggestion to use *motion* to visualize them. Motion bears a number of unique features:

- It has non zero duration so that the viewer gets some time to grasp the essence of each step.

- It naturally depicts the “flow” in the algorithm.
- It can be easily located on the, otherwise static, screen.
- It usually contains enough information to predict the outcome.

For example, exchanging the values 5 and 3 in the array in Figure 1 should be visualized by *moving* the number 3 from the last element to the place where the number 5 was before and vice versa (see Figure 6).

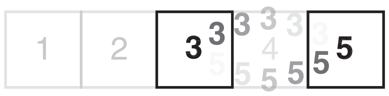

Figure 6:

Hence, we strongly recommend using motion whenever it is appropriate. Seeing motion, viewers are able to produce the correct association more quickly and easily compared to the case when they are presented just a sequence of screen shots.

Leading the Focus. When viewers observe things moving, after a short time they are able to predict the state at the end when the motion stops. Hence, motion does not only naturally lead viewers’ focus, it also provides a hint what is going to happen (e.g. that two values will be swapped). In this way, viewers are led through the whole visualization.

When a visualization uses multiple windows, the motion should lead the viewer also across different windows. For example, in Breadth First Search it is natural to show the queue where graph vertices are put before they are processed. When a vertex is being appended to the queue, a “copy” of that vertex could start moving from its original position in a graph to the end of the queue instead of just suddenly appearing there. This leads viewer’s focus in a natural way to places where changes happen.

We want to stress again that it is not enough to present an information somewhere on the screen. The viewer has to be told to look at the appropriate place. Otherwise he or she will miss it. Hence, any visualization must provide natural means – textual, visual or other – to properly lead viewer’s focus.

Algorithm Description. We assumed that the audience is not familiar with the presented algorithm yet. Thus, it is necessary to provide a proper (textual) description of it. This important issue is largely ignored by available visualization systems. Since our main goal was to *teach* algorithms, the presence of a proper description is crucial in any visualization since

- it serves as a precise description of the algorithm clarifying details that may be hard or impossible to convey by visual means;

- it enables the viewer to check and deepen the understanding of the algorithm.

Each lecturer has an own way how to present the code of an algorithm – some prefer using real programming languages, others use a Pascal-like pseudocode. In any case, for beginners it is important to present the algorithm at several levels of abstraction. A coarse sketch is suitable to explain basic ideas and design principles. Precise statements are necessary to discuss implementation details or to analyze complexity.

Hence, every visualization should be accompanied by a description of the algorithm which is available in several versions starting at a very abstract level and going down to the explanation of details. Such a description must be visible on the screen in all phases of the visualization. The rules discussed above remain valid also here. Especially, viewer's focus should be led to the "currently executed statement" so that graphical and textual information describing a single step of the algorithm is synchronous and the viewer can easily establish the connection at any time.

The viewer should be able to localize the current statement in the code quickly. Highlighting, coloring or other means could be used for this purpose. For example, to show that execution reached the for-loop in the algorithm in Figure 4, the background of the line can be set to grey (see Figure 7), marked by an arrow or typeset in a different font or color.

```
for all vertices in the neighborhood of vertex $v$ do
  ...
end
```

Figure 7:

A subtle but important issue that appears in this setting is how to make clear whether the currently marked line has already been executed or is waiting for execution. There is usually no problem with statements like for- or while-loops. But to find out the status of the line like

```
...
lower the keys of all neighbors of $v$ by 1;
...
```

Figure 8:

can prove to be tedious. Hence, to facilitate understanding of the algorithm, the viewer must be led to the right place in the code *and* must be provided an information about the execution status of that piece of code.

The presence of a description is absolutely unavoidable when visualizations are to be shared. Even though basic algorithms are rather straightforward and they do not allow too much freedom in implementation, there are still many variations possible. A visualization where the underlying code is missing is basically useless for anyone except the author. Learning details of an algorithm just from purely visual changes resembles reverse engineering which is certainly not the most efficient way of learning how things work.

4.1.3 Step 3: Show them, what you have shown them

Again, it is a habit of good speakers to provide short summaries after important sections of their talk. They give the audience a chance to reprocess and order the received information. In the same way, a good visualization should give the viewers regularly the possibility to slow down and summarize what they have seen. This helps, step by step, to build a clear understanding of the underlying algorithm.

4.2 Visualization from the Perspective of the Lecturer

In previous sections we presented guidelines for creating sound visualizations. These guidelines were mainly designed to satisfy the viewer. But before visualizations can be viewed they have to be created. Despite of this simple truth, creating visualizations is still a very time consuming task. C.H. Hundhausen showed in [Hundhausen(2002)] that it took as much as 33 hours for students to create a single visualization using *Polka*. He said: "However, my observation suggests that the graphics programming involved in creating an input-general animation shifted students' focus away from *learning the algorithm*, and toward *learning how to program graphics*."

This observation fortifies our goal to design a system that would not only support making nice visualizations but that would also allow to perform this step quickly and easily.

We switch the point of view now and look at the problem from the other side – from the position of a lecturer. To make the situation simpler, we suppose that the lecturer is the person who will have to do the programming. Even if visualizations will be reused, many places will still remain where he or she will want to incorporate own views and opinions into the existing code. Thus, the lecturer (called "user" in what follows) will be involved in the programming in the one or the other way. If a system does not allow an easy, natural and efficient programming, this fact will put a substantial burden to anyone who will want to work with it. Consequently, the acceptance will be very low – no one can afford to spend days on programming or on adapting a single visualization.

In the rest of this section we will discuss some design principles of a system that would provide for an easy creation of visualizations. The list is by no means exhaustive and we present only those principles that did not receive enough consideration in different systems yes.

At the time of this writing, a new system called *EVEGA 2* is being implemented based on this paper. Therefore, we keep the details and implementation issues short here and postpone a circumstantial discussion to a subsequent paper.

4.2.1 Language

A visualization is a process whose elementary steps change the appearance or position of objects on the screen. The main question is how to describe these transitions efficiently?

Some systems (e.g. [Rößling et al.(2000)Rößling, Schüller, and Freisleben]) designed an own interpreted language mostly consisting of statements of the form "put a green square with a side of 1cm at the point (3 cm, 4 cm)". This approach is, in principle, possible but it has numerous disadvantages. The language requires a special treatment, an own interpreter has to be written, the appearance of objects is hard-wired and it is difficult to change it. The language itself provides usually no means to perform computations and thus yet another system is needed where the algorithm itself could be implemented.

Using a real programming language yields a much more flexible working environment. What language to choose is a philosophical issue. However, since the majority of steps in visualizations are graphical and only a few solely computational, and programming graphics is more difficult than working with data structures, the chosen language should have a decent built-in support for graphics. For this reason, languages like Java having extensive graphics libraries are well suited.

4.2.2 Modular Design

A visualization system must provide the user with as much support as possible, but still leave as much freedom as possible. There should be nothing to prevent the user to change the behavior of any part of the system. These contradictory requirements can be brought together by using a modular design.

Instead of implementing a rigid system whose behavior is hard to change, we suggest to design a *toolkit* – a set of modules with flexible interfaces that the user could just plug together to obtain the desired behavior. Each module contains data structures – *models* – together with objects describing their default appearance – *views*. Moreover, it implements a versatile subsystem for exchanging messages between models and views.

Building a visualization is then carried out in three steps:

(1) Select data structures.

(2) Select appropriate views and connect them to data structures.

(3) Implement the algorithm – visual changes are triggered by sending messages to views.

Why does it make programming easy? The system provides default implementation of modules, i.e. of the data structures and the corresponding views. Thus, in the simplest case the user just takes the default implementation. Making visualization is then accomplished by sending messages that trigger appropriate actions in the views. There is no need to worry too much about programming graphics which is always the most laborious task.

Why does it make programming flexible? The modular design enables the user to replace parts that he or she is not satisfied with by own code. The interfaces ensure that replacements look to the rest of the system as "original parts".

Let us suppose that we visualize some algorithm on a graph. In order to attract viewer's attention to some vertex we want to color it red. Instead of the standard "call a procedure to paint a circle a give it the red color and the position of the vertex as parameters" we would just send a message to the view saying that the vertex should be painted with red background. We just need to name the vertex and specify the color. For the user, it is neither necessary to know how to draw a circle nor to know the current position of the vertex. Moreover, the user is saved from dealing with problems such as how to repaint things that might be visible on the top of the vertex. The system takes care of all there subtleties.

Simply speaking, modular design gives the user the possibility to change things only when necessary.

4.2.3 Hierarchical Design

The need for making changes can be further reduced by combining modular design with hierarchical structures. This brings a number of additional advantages. Hierarchies allow high flexibility at minimal cost. They may used in a wide spectrum of tasks within the system.

The advantages of object oriented languages for creating flexible systems are apparent. Models and views are implemented as classes. Employing inheritance makes changes and additions particularly easy.

But hierarchies can be useful in may different ways. Suppose, we would like to draw viewer's attention to the vertex 5 in a graph. Current visualization systems would represent this request by saying something like "draw the vertex 5 in red" and use low level procedures to make the vertex red. But the vertex 5 should not be red because of the nice red color. The real reason is that the attention of the viewer should be *attracted* to it. Thus, the correct and natural way to achieve this is not to say "draw the vertex 5 in red" but "*highlight* the vertex 5". Of course, to specify the background color of some object must be still possible, but this will happen only very rarely.

This approach has a number of positive effects.

- It facilitates keeping a visualization consistent. Since there may be different windows showing the same data structure, highlighted vertices will look everywhere the same.

- Adapting the appearance of objects to own needs is easy. Instead of changing "red background" into "blue border" everywhere in the program, the same effect can be achieved by specifying that "highlight" means "blue border" in the corresponding stylesheet for vertices.

- Last but not least, it supports writing natural code.

Thus, hierarchies, when properly used, may greatly simplify creation of visualizations. The code is adapted only at few places since changes propagate automatically. Extensions can be easily programmed since the available code is inherited.

5 Analysis of Some Available Systems

In this section we briefly discuss a few visualization systems. There are several sources containing more or less exhaustive lists of such systems (see e.g. [Khuri(2000b)]). We mention only those that are closely related to ideas presented here and analyze briefly to what extent they satisfy requirements specified in previous sections.

5.1 Animal

Guido Rößling, Berufsakademie Mannheim, [Rößling et al.(2000)Rößling, Schüller, and Freisleben].

The support for algorithm visualization (also as pseudocode) is a distinguished feature of the system *Animal*. It uses an own language to describe graphical objects – the disadvantages of this approach were discussed earlier in this paper. This fact results also in a loose connection between data and their appearance. There is no source code available.

5.2 Animated Algorithms

John Morris, The University of Western Australia.

This system written in Java has a number of nice features that partially satisfy our requirements. As one of a few systems (together with Animal), it supports visualization of the code. The user can select lines in the program that will be shown in a separate window. It uses also "high level actions" such as highlighting (instead of coloring red). There is no documentation for the code, the code itself is not modular so that incorporating changes and extensions is difficult.

5.3 EVEGA

Klaus Holzapfel, Technische Universität München, [Khuri and Holzapfel(2001)].

Java is once again the engine for this visually pleasing system that provides a large number of features. Some of them are very unique, e.g. support for sharing of algorithms via remote servers. The code is not documented, data and their representation is not separated.

5.4 Polka

John T. Stasko, Georgia Institute of Technology, [Stasko and Kraemer(1992)].

This sophisticated system written in C++ offers a variety of possibilities to create visualizations. It is one of a very few well documented systems. It implements only low level primitives to produce graphics, though, it has some distinguished features like a built-in support for motion. There is no support for visualization of the algorithm code.

6 Conclusions

Availability of numerous visualization systems paved the way for using multimedia in computer science education. Visualization of algorithms and of software in general became an inseparable part of courses at many educational institutions. Recently, studies were pursued to discover the impact of combining the traditional textual and the new graphical information.

Many problems regarding the question how to present visual information to different audiences have been solved. However, these achievements are still not reflected in the software. Making high quality visualizations is still a difficult and time consuming task.

In this paper we pointed out some of the, quite a few, missing features that must be provided by systems before high quality visualizations can be created quickly and efficiently. We also sketched the design of a new visualization toolkit that will satisfy the requirements of both viewers and lecturers.

References

[Ben-Ari(2001)] Ben-Ari, M., 2001. Program visualization in theory and practice. The European Online Magazine for the IT Professionals II (2), 8–11, http://www.upgrade-cepis.org.

[Biermann and Cole(1998)] Biermann, H., Cole, R., 1998. Comic strips for algorithm visualization, courant Institute, New York University.

[Hundhausen(2002)] Hundhausen, C. D., 2002. Integrating algorithm visualization technology into an undergraduate algorithms course: ethnographic studies of a social constructivist approach. Computers & Education 39 (3), 237–260.

[Khuri(2000a)] Khuri, S., 2000a. Designing effective algorithm visualizations, invited lecture at Program Visualization Workshop, Porvoo, Finland.

[Khuri(2000b)] Khuri, S., 2000b. List of algorithm visualization systems, http://www.mathcs.sjsu.edu/faculty/khuri/inv_links{1,2,3}.html.

[Khuri(2001)] Khuri, S., 2001. A user-centered approach for designing algorithm visualizations. The European Online Magazine for the IT Professionals II (2), 12–16, http://www.upgrade-cepis.org.

[Khuri and Holzapfel(2001)] Khuri, S., Holzapfel, K., 2001. EVEGA: An educational visualization environment for graph algorithms. In: Proceedings of the 6th Annual Conference on Innovaton and Technology in Computer Science Education, ITiCSE 2001. ACM Press, New York.

[Naps(2001)] Naps, T. L., 2001. Incorporating algorithm visualization into educational theory: A challenge for the future. The European Online Magazine for the IT Professionals II (2), 17–21, http://www.upgrade-cepis.org.

[Rößling et al.(2000)Rößling, Schüller, and Freisleben] Rößling, G., Schüller, M., Freisleben, M., 2000. The Animal algorithm animation system. In: Proceedings of the 5th Annual Conference on Inovation and Technology in Computer Science Education, ITiCSE 2000, Helsinki, Finnland. pp. 37–40.

[Stasko and Kraemer(1992)] Stasko, J. T., Kraemer, E., Jun. 1992. A methodology for building application-specific visualizations of parallel programs. Tech. Rep. GIT-GVU-92-10, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA.