Key Decisions in Adopting Algorithm Animation for Teaching

Guido Rößling
FG Telecooperation, FB 20, Darmstadt University of Technology, Alexanderstrasse 6, D-64283 Darmstadt, Germany

roessling@informatik.tu-darmstadt.de

Key words: Algorithms, Distance Learning, Education, e-learning, Learning Materials

Abstract: Algorithm Animation is becoming increasingly popular with several educators. However, certain key questions have to be addressed before an informed decision on a given system can be made. This paper identifies several such questions and provides a brief overview of some algorithm animation systems. The key decisions are grouped in four categories: system type and animation generation approach; display properties; import and export facilities; and didactical requirements. After considering these requirements, selecting a particular system should be easier for educators.

1. INTRODUCTION

Algorithm Animation (AA) has received increasing interest over the last few years within the education community. This is reflected in the number of publications at educational conferences (such as the yearly ACM SIGCSE and ITiCSE conferences), the establishment of an international Program Visualization Workshop in 2000 and a Dagstuhl Seminar in May 2001. However, many educators still wonder what algorithm animation is and how it could help them in their teaching.

Simply put, algorithm animation is the dynamic visualization of the processes that make up algorithms and data structures. Thus, any tool that

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35663-1_34

T. J. van Weert et al. (eds.), *Informatics and the Digital Society*

illustrates the execution of a given algorithm can be counted as AA. Today many different systems exist. Educators interested in employing AA therefore have to make an informed decision to adopt a given system, or decide to develop their own. However, there are vast differences between the available tools. In this paper, we will examine typical examples of the different approaches taken by a restricted selection of AA systems. We hope that the survey is helpful as a guideline for deciding on a given system. Ideally, it will also help generate interest in AA from educators who are still unsure whether or not AA may assist them in their teaching.

The basic reference for AA is the book "Software Visualization" edited by John Stasko et al (1998a). This book includes two elaborate taxonomies for classifying AA systems. However, these taxonomies do not focus on the factors that influence the adoption of a particular AA system for teaching. This paper focuses on outlining central aspects that should be taken into account when deciding whether or not AA should be adopted into teaching.

2. KEY DECISIONS

We can classify the key decisions that must be made into five different areas - the type of system, how animations are generated or edited, content display properties, import and export facilities, and didactical requirements. A complete discussion of all the requirements that can be placed on AA systems is beyond the scope of this paper - it focuses instead on outlining the central issues that need to be addressed.

2.1 AA System Type and Content Generation

The type of system represents the basic approach followed by a given system. We regard four different user roles - *programmer, developer, visualiser* and *user* (Price et al 1998). The *programmer* role implements a given algorithm without regard for animation purposes. In fact, the programmer may be unaware of any plans for future animation of the code. The *developer* role covers the design and construction of the AA system used to portray the animation content. Content generation and editing is the task of the *visualiser* role. Finally, the *user* role represents the end-user who studies the animation and possibly interacts with it.

The AA system can be classified as belonging to one of six different types according to the content generation method employed. *Topic-specific* systems are implemented for a certain target area, for example graph algorithms. They may provide very good support for this field of expertise, but are often not usable outside their focus. We examined the applications

RLE, *Quadtree* and *JPEG* which were representative of this category of systems. They portray the properties of the image compression and encoding approaches identified by Khuri and Hsu (2000). The packages provide verbose explanations for the topic area but each is "hard-wired" to a particular problem and thus cannot easily be reused. The content is generated from the user input, so that the visualiser role is not actually able to generate an animation according to his or her preferences.

Systems that allow content generation within a Graphical User Interface (*GUI*) give the greatest degree of design freedom to the visualiser. There are two main drawbacks. First, all of the content must be generated manually which is very time-consuming. Secondly, as the content is typically generated independently of the algorithm that is to be illustrated, the visualiser is responsible for ensuring that the algorithm is portrayed correctly. On the other hand, manual generation also allows abstract displays that are based on pseudo-code, use metaphors, or focus only on certain aspects of the algorithm. The selection of AA systems that incorporate visual generation is rather limited and, apart from presentation tools such as PowerPoint and Star Impress, the main system currently in use is ANIMAL (Rößling and Freisleben 2000).

API-based generation uses method invocations from a special library for visualizing the content of the algorithm. The necessary method invocations are usually embedded in a modified version of the underlying algorithm. A new animation is therefore generated simply by restarting the algorithm. One of the best known of the older animation systems, *XTANGO*, employed a special C-based API (Stasko 1998b). API-based generation can be very elegant. However, this elegance depends on the features offered by the API. The method invocations embedded in the source code modify the running time and may also result in side effects. The visualiser has to be skilled in the programming language underlying the API, and is typically restricted to algorithms implemented in the same underlying programming language. Additionally, the animation is usually performed immediately, and there may be no provision for exporting to a file so that the animation could be reused.

Scripting-based generation parses a set of commands to generate the animation content. Typically, the commands are gathered in a file, in this case allowing reuse of the animation content. As most scripting languages are relatively straightforward in their notation the visualiser can also modify the commands using any text editor. The commands can be generated manually by simply typing them in an editor or can be generated from the algorithm, typically by sending the appropriate commands to the standard output or error terminal and redirecting this output to a file. Two typical representatives of scripting-based systems are *JAWAA* (Pierson and Rodger 1998) and *JSamba* (Stasko 1998b). The scripting languages employed by the

systems are similar but incompatible. While comparable in their general approach, there are interesting differences between the systems. For example, JAWAA is specifically geared for Computer Science content, providing special commands for common data structures and operations thereon, such as trees or stacks. JSamba is more general, requiring the user to assemble stacks manually from base components. However, JSamba provides a simple zooming function that can be used by the visualiser to focus the user's attention.

Declarative generations embed special logical predicates into the algorithm, usually within comments. The comments are extracted and evaluated by special preprocessors or full-fledged virtual machines. Alternatively, a modified compiler can be used that replaces the predicates with appropriate pieces of code. Leonardo (Crescenzi et al 2000) is one example system that follows declarative visualization.

Finally, some systems use *code interpretation* to visualise the algorithm. One of the best known systems is *ZStep 95* (Liebermann and Fry 1998). It evaluates and visualises LISP code. It is interesting to note that no visualiser interaction is required for generating the animation, as this is performed automatically. This also means that visualisers cannot tailor the appearance of the animation to their preferences.

In general, there is no single "best" method of animation generation and editing - it really depends on the preferences and skills of the visualiser. *Manual generation* offers the greatest flexibility of presenting the content - however, it is easily the most time-intensive method. Direct *interpretation* of the underlying code causes the least work for the visualiser - however, the appearance can usually not be adjusted. *API-based* and *declarative* generations are "cleaner" ways for embedding visualization generation statements into the underlying algorithm than is the case for *scripting*; however, the visualiser needs a certain amount of familiarity with the underlying programming language and logic to use the first two techniques. Finally, *topic-specific* systems are often highly specialized, but may restrict the freedom of the visualiser with regard to adjustment of the display. Additionally, they are usually ill suited for use outside their specific area of experience. Alas, most systems do not combine more than a single generation approach, with the exception of ANIMAL (Rößling and Friesleben 2000). It offers GUI, scripting and API-based animation generation.

2.2 Animation Display Criteria

To address different needs of the user, the animation display should address several issues in its display front-end. First of all, the animation canvas should be resizable to allow it to fit a smaller or larger display (a

notebook instead of a desktop PC). For presentations in larger lecture halls, the content should also be freely scalable. Of the systems discussed here, only ANIMAL and JSamba offer zooming - the former within the display GUI, the latter only by animation commands. Thus, the presenter of an animation cannot easily adapt the zoom scale to one appropriate for presentation using a beamer.

The second issue regards the controls for steering the behaviour of the animation display. Ideally, the full functionality of a video player should be provided - including fast forward and reverse playing. Currently, the number of systems that offer "real" reverse playing is very limited, due to the difficulty in implementing free reverse playing. In 2001 researchers stated that efficient reverse animation playing and rewinding can be considered one of the foremost "open questions" in AA (Anderson and Naps 2001). Two representative tools that offer this functionality are ANIMAL and ZStep 95 (Liebermann and Fry 1998). Most other systems are restricted to offering *play, pause* and a *step forward* functionality that shows the next state without animating the content.

The visualiser should also be able to specify how a given animation step is reached from its predecessor. In some systems, such as JAWAA and JSamba, the next step is automatically executed after the current step has completed execution. Optionally, the visualiser may issue an appropriate command to pause the display. The image compression packages by Khuri and Hsu (2000) show the next state only after a user interaction. ANIMAL combines both approaches: for each animation step the visualiser can define if the next step is shown automatically after a certain delay or only after a user interaction. In addition to the *pause* button offered by many systems, ANIMAL also offers a "slide show" mode. This links all steps with a predefined delay. The mode is automatically left when the pause button is depressed.

2.3 Animation Import and Export

Most current animation systems offer no support for exporting the animation content to a different format or importing foreign formats. Some systems, such as the code interpretation-based WinHIPE (Velázquez-Iturbide and Presa-Vásquez 1999) offer at least an export to graphic formats such as bitmaps or GIF images. The lack of import and export facilities in most systems is surprising, as the provision of these facilities is usually not very difficult, and provides powerful incentives for users to adopt the system due to the possibility of content reuse. ANIMAL again is one of the exceptions to this rule, offering export to various image formats such as JPEG and PNG, XML streams, and Quicktime videos. Some of the export

formats require special libraries that have to be installed separately. Therefore, the export formats are treated as extensions and not provided in the basic distribution.

2.4 Didactical requirements

Several didactical requirements for improved retention have been discussed recently in the AA community. One of the foremost of these is interactive prediction, as incorporated in the *JHAVÉ* system (Naps 2001). Here, the user is asked at fixed points to predict what the next operation shown by the AA system should be. In *JHAVÉ*, the prediction is performed as a multiple-choice quiz. Some other systems such as *PILOT* (Bridgeman 2000) let the user actually "perform" the operation by interacting with the display. This type of interaction requires a great deal of context awareness on the part of the system, and is therefore usually limited to topic-specific systems. The main benefit of interactive prediction, and generally of AA, is the raised interest and motivation of the students. Recent studies disagree on the learning effect inferred by prediction or AA in general. Obviously the learning effect is also largely dependent on the way the content is presented and the way that users can interact with it.

Naps argues for the value of incorporating links to external documentation for the displayed content (Naps 2001). The documentation is typically encoded in HTML and should provide additional information about the current display and the algorithm in general. Naps differentiates between fixed or "static" documentation, *algorithm-sensitive* documentation that is aware of the current stage of the displayed content, and *dynamic* documentation which is algorithm-sensitive and also incorporates the concrete values of the current animation display.

3. CONCLUSION

Many educators are willing to employ algorithm animation in their lectures, but are not sure how to do it. One large part of this uncertainty is the lack of knowledge about the available systems and how they can be compared and evaluated. It is difficult to obtain a full list of available animation systems as their number was estimated at 150 in 1998 and has risen since then (Price et al 1998). However, there are some good resources which give information about AA and AA systems:

- http://www.cs.hope.edu/~alganim/ccaa/index.html
- http://www.cs.cofc.edu/~mccauley/edlinks/
- http://www.animal.ahrgr.de

In this paper we have focused on presenting some of the key decisions that educators should consider when they decide to adopt AA for their courses.

The foremost consideration is the approach used for generating new animations. This has a large effect on the system's usability and depends on the educator's skills and preferences. Other issues relating to the display of the animation content, import and export, and didactical requirements have been discussed. The breadth of the topic is so wide that several issues could not be covered in this paper. We trust that our analysis will prompt readers to think deeply about the requirements for, and of AA systems.

REFERENCES

Anderson, J. M. and Naps, T. (2001) A Context for the Assessment of Algorithm Visualization Systems as Pedagogical Tools. In *Proceedings of the First International Program Visualization Workshop*, 2000. E. Sutinen et al (eds.), University of Joensuu Press, Joensuu, Finland. pp. 121-130.

Bridgeman, S. (2000) PILOT: An Interactive Tool for Learning and Grading. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, 2000. ACM Press, New York. pp. 139-143.

Crescenzi, P., Demestrescu, C., Finocchi, I. and Petreschi, R. (2000) Reversible Execution and Visualisation of Programs witth LEONARDO. *Journal of Visual Languages and Computing*, Vol. 11, No. 2, April, pp.125-150.

Khuri, S. and Hsu, H-C. (2000) Interactive Packages for Learning Image Compression Algorithms. In *Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education* (ITiCSE 2000), J. Tarhio (ed.), ACM Press, New York. pp. 73-76.

Liebermann, H and Fry, C. (1998) ZStep 95: A Reversible, Animated Source Code Stepper. In *Software Visualization – Programming as a Multimedia Experience*. J. Stasko et al (eds.), MIT Press. pp. 277-292.

Naps, T. (2001) Incorporating Algorithm Visualization into Educational Theory: A Challenge for the Future. *Informatik / Informatique Special Issue on Visualization of Software*, April 2001. pp. 17-21.

Price, B. et al (1998) An Introduction to Software Visualization. In *Software Visualization – Programming as a Multimedia Experience*. J. Stasko et al (eds.), MIT Press. pp. 3-27.

Pierson, W. and Rodger, S. (1998) Web-based Animation of Data Structures Using JAWAA. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*. J. Lewis (ed.), ACM Press, New York. pp. 267-271.

Rößling, G. and Freisleben, B. (2000) The Animal Algorithm Animation Tool. In *Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education* (ITiCSE 2000), J. Tarhio, (ed.), ACM Press, New York. pp. 37-40.

Stasko, J. et al (1998a) *Software Visualization – Programming as a Multimedia Experience*. MIT Press.

Stasko, J. (1998b) Smooth Continuous Animation for Portraying Algorithms and Processes. In *Software Visualization – Programming as a Multimedia Experience*. J. Stasko et al (eds.), MIT Press. pp. 103-118.

Velázquez-Iturbide, J. Á. And Presa-Vázquez, A. (1999) Customization of Visualizations in a Functional Programming Environment. *Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference*, IEEE Press, New York, pp. 12b3 22-28.