Chapter 1
Algorithm Animation

Introduction

Andreas Kerren[1] and John T. Stasko[2]

[1] FR 6.2 Informatik,
Saarland University,
PO Box 15 11 50, D-66041 Saarbrücken, Germany.
kerren@cs.uni-sb.de,
http://www.cs.uni-sb.de/~kerren/

[2] College of Computing / GVU Center,
Georgia Institute of Technology,
Atlanta, GA 30332-0280, USA.
stasko@cc.gatech.edu,
http://www.cc.gatech.edu/~john.stasko/

An *algorithm animation* (AA) visualizes the behavior of an algorithm by producing an abstraction of both the data and the operations of the algorithm. Initially it maps the current state of the algorithm into an image, which then is animated based on the operations between two succeeding states in the algorithm execution. Animating an algorithm allows for better understanding of the inner workings of the algorithm, furthermore it makes apparent its deficiencies and advantages thus allowing for further optimization.

Price, Baecker and Small [63] distinguish between algorithm animation and program animation. The first term refers to a dynamic visualization of the higher-level descriptions of software (algorithms) that are later implemented in software. The second term refers to the use of dynamic visualization techniques to enhance human understanding of the actual implementation of programs or data structures. Price, Baecker, and Small define both areas of study to collectively be a part of *Software Visualization* (SV). Here in this introduction we loosen this distinction, i.e., the discussed systems can be subsumed by the terms algorithm and program animation.

Two extensive anthologies about software visualization were published in 1996 and 1998 [33,78]. Both provide overviews of the field. The latter one also contains revised versions of some seminal papers on classical algorithm animation systems as well as educational and design topics. Other published articles provide summaries of different aspects of algorithm animation in particular, including taxonomies [10], the use of abstraction [22], and user interface issues [39]. In this

S. Diehl (Ed.): Software Visualization, LNCS 2269, pp. 1–15, 2002.

introduction we first provide a short summary of the historical development of software visualization, especially algorithm and program visualization. In this context we concentrate on systems that introduced new concepts.

Next, we survey some newer systems on the basis of four concepts or dimensions: specification technique, visualization technique, language paradigm, and domain specific animations. Because these systems have been developed more recently, they usually were not discussed in the aforementioned anthologies. Due to space limitations, however, the following sections only describe some representative systems, with a particular focus on systems that are presented in the seven papers of this chapter.

1 Classical Systems and Concepts

Knowlton's movie [47] about list processing using the programming language L6 was one of the first experiments to visualize program behavior with the help of animation techniques. Other early efforts often focused on aiding teaching [1,44] including the classic "Sorting Out Sorting" [3,2] that described nine different sorting algorithms and illustrated their respective running times.

Experiences with algorithm animations made by hand and the wide distribution of personal computers with advanced graphical displays in the 1980's led to the development of algorithm animation systems. The well known system Balsa [19,8] introduced the concept of *Interesting Events* (IE's) and with it the binding of several views to the same state. In this approach the key points of the program are annotated with IE's by the visualizer (the person who specifies the visualization). When those IE's are reached during execution, an event, parameterized with information about the current program state, is sent to the views. The successor Balsa II [9] was extended with step and break points and a number of other features. In Zeus [11], Cat [14], and the later Java based system Jcat [18] the views were distributed on several workstations.

The system Tango [72] implemented the path-transition paradigm [73,77] that permitted smooth and concurrent animations of state transitions. In its successor Polka [79], these properties were revised to facilitate easier design of concurrent animation actions. As a front-end of Polka, an interactive animation interpreter called Samba [75] (including the later Java based Jsamba) was developed. Samba consisted of a number of parameterized ASCII commands that performed different animation actions. Thus, programs written in any programming language could be animated simply by having them output Samba commands at interesting event points.

The system Pavane [67] was noteworthy in exploring a declarative animation paradigm in which program states were mapped to visualization states. The animation designer simply created this mapping initially, then the program ran and the appropriate mappings were maintained.

A number of other noteworthy algorithm visualization systems and tools have been developed over the years. Some of the earlier efforts include systems in

Smalltalk [52,32], the ALADDIN system [45,42], the MOVIE system [6], animations for algorithms textbooks [37,38], and the GAIGS system [56].

Recently, a number of new systems have been introduced. Many of these newer systems were presented at workshops and conferences, including the GI workshop SV'2000 [27], the First International Program Visualization Workshop 2000 [82] and the Dagstuhl Seminar on Software Visualization 2001 [28]. In the next four sections, we briefly describe a few of the newer systems, as well as noteworthy earlier systems, with respect to four important dimensions: specification technique, visualization technique, language paradigm, and domain-specificity.

2 Specification Technique

An important practical task in creating algorithm visualizations is to specify how the visualization is connected or applied to the algorithm. SV researchers have developed a number of approaches to this problem. In this section we will examine some of the different approaches and the systems that use the approaches. Note that some of the systems could be classified in several categories.

2.1 Event Driven

The *interesting event* approach was pioneered by BALSA [19,8] and has been used in many algorithm animation systems including its successor ZEUS [11]. As mentioned above, the visualizer identifies key points in the program and annotates these with IE's. Whenever an IE is reached during execution, a parameterized event is dispatched to the views.

The event-based framework GANIMAL [31] offers some new features like alternative interesting events, alternative code blocks, mixing of post-mortem and live/online algorithm animation, visualization control of loops and recursion, etc. Annotations are provided by the GANILA language, which are compiled into Java. The generated code allows the association of meta-information (settings) with each program point of the algorithm. Consequently a graphical user interface can be used to change these settings at runtime of the animation.

The ANIMAL system [69,68] also utilizes an event-based approach and provides a number of advanced features for animation presentation including dynamic flexibility of animation mappings, reverse execution, internationalization of animation content, and flexible import and export capabilities.

2.2 State Driven

An alternative approach is to specify a mapping between program and visualization states, usually constructed by the visualizer before program execution. As discussed above, the PAVANE system was an early adopter of this technique [66, 65]. The declarative approach is also utilized by some newer systems. LEONARDO [23,24] is an integrated environment for developing, executing and animating C programs. Visualizations are specified by adding declarations written in ALPHA,

a declarative language, to the C program. LEONARDO also supports full reversible execution by using a specialized virtual machine that executes the compiled C program.

DAPHNIS is an algorithm animation system based on the use of data flow tracing. Some aspects of abstraction in the visualization are produced fully automatically, but to prepare an animation, it is necessary to supply an external configuration script that specifies the graphical representation and rules of translation for all the variables to be visualized. To achieve spatial or temporal suppression of unimportant information, a special kind of the Petri net formalism is applied to describe the process of algorithm execution. The DAPHNIS system as well as its theoretical model are discussed further in this chapter [36].

Demetrescu, Finocchi, and Stasko provide a direct comparison of both the *interesting event* and *state mapping* approaches in this chapter [25], identifying some scenarios where one might be preferable to the other.

2.3 Visual Programming

Another technique for specifying algorithm animations is the use of visual programming techniques. Visual programming (VP) seeks to make programs easier to specify by using a visual notation for the actual program commands and statements. As a whole, VP is considered to be distinct from SV [63], but such a graphical notation itself is a kind of statical code/data visualization.

One well-known project to embed animation capabilities into a visual programming language (VPL) is the declarative VPL FORMS/3 [20] in which animation is done by maintaining a network of one-way constraints. The developers of this language integrated an extension of the path-transition paradigm into their language, resulting in a unique approach to algorithm animation, e.g. a seamless integration of algorithm animation into the language and on-the-fly exploratory programming of an algorithm animation.

An area related to VP is programming by demonstration (PbD). In PbD, a person demonstrates an example or an operation of a task, and the PbD-system infers a program for this task. DANCE [71,76] is a PbD-interface to the TANGO system. After the user demonstrates an animation scenario in a direct manipulation style graphical editor, the DANCE system generates ASCII code that specifies the animation. This code is then used as input to TANGO.

2.4 Automatic Animation

Perhaps the simplest way to specify an animation, at least for the algorithm developer, is to have the animation automatically generated. Total, automatic creation of algorithm animations is extremely difficult however [10], so systems have provided differing levels of automation to specify algorithm animation. Because automated animation creation requires little or no effort on the programmer's part, this approach is very well-suited to debugging [54].

An early system in this area, UWPI [43], provided automatic animation by using a small expert system that chose the visualization for the data structures

and operations of an algorithm by attempting to infer the abstract data types used in the program. The system could display abstractions of higher-level data structures, even though it did not truly "understand" them.

JELIOT is a family of program animation environments some of which support the semi-automated paradigm by allowing users to define the visual semantics of program structures or to select the most adequate ones. One system was exclusively developed for novice programmers. It supports fully automatic animation, and does not allow any customization of the animation. Ben-Ari, Myller, Sutinen, and Tarhio give an overview of the JELIOT family and discuss some empirical evaluations of some of the systems in this chapter [5].

Another technique that fits this category is the use of special pseudo-code languages in which programmers implement their code, and then the animation is automatically produced. ALGORITHMA 99 [21] is such an example system.

3 Visualization Technique

One of the most important tasks in SV is the design of the graphical appearance of a visualization or animation. The display design must address a number of different issues, e.g., what information should be presented, how should this be done, should there be a focus on the important elements, and so on. Brown and Hershberger give a good overview of fundamental techniques on this topic [12]. In this introduction we discuss three aspects of visualizing algorithms that have received much attention lately: 3D Algorithm Animation, Auralization, and Web Deployment.

3.1 3D Algorithm Animation

There may be several reasons for integrating 3D graphics into an algorithm animation system. The third dimension can be used for capturing time (history), uniting multiple views, and displaying additional information [16]. Both the systems POLKA [81] and ZEUS [15] were extended with 3D graphics versions. Brown and Najork further integrated their earlier work on the platform-dependent ZEUS3D into the JCAT system. With the resulting Java-based system, 3D animations could be run in any standard web browser [17,55]. The 3D animations were implemented using the object-oriented, scene-graph based graphics library Java3D (plugin). In the GASP system, Tal and Dobkin explored 3D animations of computational geometry algorithms also [83]. They created a library of geometric data types including operations that were furnished with animation instructions.

3.2 Auralization

In SV, audio can be used to reinforce and replace visual cues, to convey patterns, to identify important events in a stream of data, and to signal exceptional conditions [13]. Recently the mapping of information to musical sound using

parameters such as rhythm, harmony, melody or special leitmotifs has been studied.

CAITLIN [85] is a preprocessor for Pascal which allows a user to specify an auralization for each type of program construct (e.g. a `FOR` statement) on the basis of a hierarchical leitmotif design. CAITLIN does not allow auralization of data, however. Empirical studies [86] of this system show that novice programmers could interpret the musical auralizations, that musical knowledge had no significant effect on performance, and that, in certain circumstances, musical auralizations can be used to help locate bugs in programs.

The musical data sonification toolkit MUSE [51] provides flexible data mappings to musical sounds. The data can come from any scientific source. It is written for the SGI platform and supports different mapping types of data to sound, like timbre, rhythm, tempo, volume, pitch and harmony.

A similiar system is FAUST [88], a framework for algorithm understanding and sonification testing. It allows simple mappings of algorithm events to sound parameters and requires programmers to manually tag events in their algorithms. Furthermore, the interested programmer can easily change sound synthesis algorithms and add new features and attributes to these algorithms.

3.3 Web Deployment

With the growing use of the World Wide Web as a generic application and display platform, a number of recent algorithm animation systems have focused on delivery of animations over the Web. The JSAMBA and JCAT systems mentioned earlier are two examples. Other systems presenting animations over the Web include JHAVE [57], the SORT ANIMATOR [26], JELIOT [41], and JAWAA [62].

4 Language Paradigm

Different language paradigms may need different abstractions and entities to be visualized, due to their unique styles of computation and methods for problem solving. The survey by Oudshoorn, Widjaja, and Ellershaw [59] analyses the visualization requirements for a variety of programming paradigms and gives a simple taxonomy of program visualization. In the following section we consider the most important language paradigms and illuminate some example systems.

4.1 Imperative Programming Languages

The imperative paradigm is based on the procedural approach to solve problems. From this point of view, the developer of an algorithm animation has to find abstractions of variables, data structures, procedures/functions and control structures. The BALSA system [9] exemplifies this paradigm.

4.2 Functional Programming Languages

The most significant abstractions for functional languages are functions and data structures. A textual browser to view the trace of the evaluation of a lazy functional language is discussed in [87]. The system facilitates navigating over a trace and it can be used as a debugging tool or as a pedagogical aid in understanding how lazy evaluation works.

The KIEL System [7] is an interactively controlled computer system for the execution of first-order functional programs written in a simple subset of Standard ML. In contrast to the prior discussed system, it offers ways to visualize the evaluation process.

A formal model of traversing graphical traces of lazy functional programs is introduced by Foubister [35]. This model provides the visual representation of graph reduction by template instantiation, and solves some problems in displaying the reduction, e.g., the large size of the graphs or their planarity.

4.3 Object-Oriented Programming Languages

The object-oriented paradigm has much in common with the imperative language paradigm. As a consequence, visualizations of object-oriented programs typically consider abstractions of objects, including inter-object communication, in addition to the above-mentioned abstractions for imperative languages.

One of the papers in the following chapter [58] deals with solutions for the endemic problem of aliasing within object-oriented programs, i.e., a particular object can be referred to by any number of other objects via its reference. This fact may be unknown to the algorithm animation system and can cause problems for animations. The paper discusses analysis of the program to determine the extent of aliasing as well as a visualization of ownership trees of objects in Java programs.

SCENE [48] automatically produces scenario diagrams (event trace diagrams) for existing object-oriented systems. This tool does not provide visualization of the message flow in an object-oriented program, but by means of an active text framework it allows the user to browse several kinds of associated documents, such as source code, class interfaces or diagrams, and call matrices.

4.4 Logical Programming Languages

In logic programs, the interesting abstractions are clauses and unification. One of the classic systems for visualizing logic programs is the Transparent Prolog Machine TPM [34]. It uses an AND/OR tree model of the search space and the execution of the logic program is shown as a DFS search. Another system for presenting logic programs was discussed by Senay and Lazzeri [70].

5 Domain Specific Animations

The search for adequate abstractions of the specific properties of a particular domain, e.g., realtime algorithms, can be a challenge when animating algorithms in

such a focused domain. To find such abstractions, knowledge regarding the types of objects and operations that are dominant in the special domain is necessary. Often, general algorithm animation systems can be used to build domain-specific animations, but the effort can be extensive and much more involved than that required for a more narrowly-focused system.

Tal presents a conceptual model for developing algorithm animation systems for constraint domains in this chapter [84]. She illuminates the practical implementation of this model on the basis of a few example systems, e.g., the GASP system, mentioned earlier.

5.1 Computational Geometry

In computational geometry, the task of finding abstractions of the data can be relatively easy if the program data contains positional information. Hence, it can be displayed without complicated transformation.

A generic tool for the interactive visualization of geometric algorithms is GEOWIN discussed by Bäsken and Näher in this chapter [4]. It is a C++ data type which can be interfaced with algorithmic software libraries such as LEDA.

The EVEGA system [46] is a Java-based visualization environment for graph algorithms and offers a set of features to create and edit graphs, to display visualizations and to perform comparisons of different algorithms. Furthermore, it supports a relatively straightforward implementation of new algorithms by class extension.

5.2 Concurrent Programs

The animation of concurrent programs, which are typically very complex and large, must address a number of problems with regard to data collection, data display and program execution. Some problems encountered are inherently visual, e.g., a cycle in a resource allocation graph corresponds to a deadlock situation. Other problems can be a non-deterministic occurrence of a bug in program execution, which requires a clever visualization of the concurrent program. For an overview of systems for visualizing concurrent programs, see [49].

The event-based PARADE environment [74] supports the design and implementation of animations of parallel and distributed programs. Interesting events may be received via program calls, through pipes or read from a file, similar to the aforementioned SAMBA interpreter. A particular component of the system gathers the events for each processor or process and allows the user to manipulate the order of these events, e.g. chronologically or logically [50]. The POLKA animation system is used in PARADE to build the graphical views.

VADE [53] is a client-server based system for visualizing algorithms in a distributed environment. It provides libraries that facilitate the automatic generation of visualizations, and supports web page creation for the final animation. Furthermore, VADE offers synchronization methods that maintain consistency.

5.3 Real-Time Animation

Some domains, such as network protocols, need to represent exact timing relationships in the underlying program or algorithm. Polka-Rc [80] is an extension of Polka with features for real-time animation, i.e., animation actions of precise timings, initiations and durations. It also provides a flexible multiprocess mapping between program and visualization. More precisely in Polka-Rc the program and its animation run as separate processes and communicate via sockets.

The Jotsa system [64] is a Java package for performing interactive web-based algorithm animations (especially for network protocols). It supports exact time animation, multiple independent synchronized views, panning, zooming and linking of collections of objects. In addition it has facilities for animation of user-defined event-driven and timer-driven simulations of network protocols.

5.4 Computational Models

Another domain of interest to algorithm animators is the animation of computational models of formal languages, sets, relations, and functions. These models are typically for mathematical reasoning, not for programming of real hardware and real applications.

Jflap [40] is a tool for creating and simulating several kinds of automata, i.e. finite automata, pushdown automata and Turing machines, and for converting representations of languages from one form to another. Jflap is written in Java and has been used both in and outside of the classroom.

Diehl and Kerren [29] discuss how generation of visualizations of computational models and the visualization of the generation process itself increase exploration. Four approaches of increased exploration in formal language theory and compiler design are introduced and each approach is exemplified by an implemented system. As an example of such a system, the authors characterize GaniFa [30], a Java applet for the visual generation and animated simulation of finite automata.

5.5 Animation of Proofs

A relatively unexplored area is the visualization of proofs in theoretical computer science education. An example is Scapa [60,61] a system for the animation of structured calculational proofs. This system generates both an HTML document (with the help of a converter) and a Java file from a proof written in LaTeX. The visualizer has to extend and modify these files. The proof animation is finally created by using an extended version of the Lambada tool, which is a Java-based reimplementation of Samba.

6 Conclusion

Much progress has been made in the field of Algorithm Animation since the first films motivating the field, such as *Sorting Out Sorting*, were made. In this chapter

introduction, we have highlighted a number of the landmark systems that have been developed in the area, plus we have surveyed some new developments. A brief introduction like this, however, is in no way a comprehensive overview of the field. We encourage the reader to use this introduction and the articles in this chapter as a starting point for exploring other research and creating new systems and techniques.

References

1. R. Baecker. Towards Animating Computer Programs: A First Progress Report. In *Proceedings of the Third NRC Man-Computer Communications Conference*, 1973.
2. R. Baecker. Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 24, pages 369–381. MIT Press, Cambridge, MA, 1998.
3. R. Baecker (with assistance of Dave Sherman). Sorting out Sorting. 30 minute color film (distributed by Morgan Kaufmann Pub.), 1981.
4. Matthias Bäsken and Stefan Näher. GeoWin A Generic Tool for Interactive Visualization of Geometric Algorithms. In *Proceedings of Dagstuhl Seminar on Software Visualization, 2001.*
5. Mordechai Ben-Ari, Niko Myller, Erkki Sutinen, and Jorma Tarhio. Perspectives on Program Animation with Jeliot. In *Proceedings of Dagstuhl Seminar on Software Visualization, 2001.*
6. J. L. Bentley and B. W. Kernighan. A System for Algorithm Animation. *Computing Systems*, 4(1), Winter 1991.
7. R. Berghammer. KIEL: A Program for Visualizations of the Evaluation of Functional Programs (in German). In *S. Diehl and A. Kerren, editors: Proceedings of the GI-Workshop "Software Visualization" SV2000*, May 2000.
8. M. H. Brown. *Algorithm Animation.* MIT Press, 1987.
9. M. H. Brown. Exploring Algorithms with Balsa-II. *Computer*, 21(5), 1988.
10. M. H. Brown. Perspectives on Algorithm Animation. In *Proceedings of the ACM SIGCHI '88 Conference on Human Factors in Computing Systems*, pages 33–38, Washington D.C., May 1988.
11. M. H. Brown. ZEUS: A System for Algorithm Animation and Multi-View Editing. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, pages 4–9, Kobe Japan, October 1991.
12. M. H. Brown and J. Hershberger. Fundamental Techniques for Algorithm Animation Displays. In John T. Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization.* MIT Press, 1998.
13. M. H. Brown and J. Hershberger. Program Auralization. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 10, pages 137–143. MIT Press, 1998.
14. M. H. Brown and M. Najork. Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom. In *Proceedings of the 1996 IEEE International Symposium on Visual Languages*, Boulder, CO, 1996.
15. M. H. Brown and M. A. Najork. Algorithm Animation Using 3D Interactive Graphics. In *Proceedings of the 1993 ACM Symposium on User Interface Software and Technology*, pages 93–100, Atlanta, GA, November 1993.

16. M. H. Brown and M. A. Najork. Algorithm Animation Using Interactive 3D Graphics. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 9, pages 119–135. MIT Press, 1998.
17. M. H. Brown and M. A. Najork. Three-Dimensional Web-Based Algorithm Animations. Technical Report 170, Compaq Systems Research Center, July 2001.
18. M. H. Brown and R. Raisamo. JCAT: Collaborative Active Textbooks Using Java. In *Proceedings of CompuGraphics'96*, Paris, France, 1996.
19. M. H. Brown and R. Sedgewick. A System for Algorithm Animation. In *Proceedings of ACM SIGGRAPH'84*, Minneapolis, MN, 1984.
20. P. Carlson, M. Burnett, and J. Cadiz. A Seamless Integration of Algorithm Animation into a Declarative Visual Programming Language. In *Proceedings Advanced Visual Interfaces (AVI'96)*, 1996.
21. A.I. Concepcion, N. Leach, and A. Knight. Algorithma 99: An Experiment in Reusability and Component-Based Software Engineering. In *Proceedings of the 31st ACM Technical Symposium on Computer Science Education (SIGCSE 2000)*, pages 162–166, Austin, TX, February 2000.
22. K. C. Cox and G.-C. Roman. Abstraction in Algorithm Animation. In *Proceedings of the 1992 IEEE Workshop on Visual Languages*, pages 18–24, Seattle, WA, September 1992.
23. C. Demetrescu and I. Finocchi. A Technique for Generating Graphical Abstractions of Program Data Structures. In *Proceedings of the 3rd International Conference on Visual Information Systems (Visual'99)*, LNCS 1614, pages 785–792, Amsterdam, 1999. Springer.
24. C. Demetrescu and I. Finocchi. Smooth Animation of Algorithms in a Declarative Framework. *Journal of Visual Languages and Computing*, 12(3):253–281, June 2001.
25. Camil Demetrescu, Irene Finocchi, and John Stasko. Specifying Algorithm Visualizations: Interesting Events or State Mapping? In *Proceedings of Dagstuhl Seminar on Software Visualization, 2001.*
26. H. L. Dershem and P. Brummund. Tools for Web-based Sorting Animations. In *Proceedings of the 29th ACM Technical Symposium on Computer Science Education (SIGCSE 98)*, pages 222–226, Atlanta, GA, February 1998.
27. Eds.: S. Diehl and A. Kerren. Proceedings of the GI-Workshop "Software Visualization" SV2000 (in German). Technical Report A/01/2000, FR 6.2 - Informatik, University of Saarland, May 2000. `http://www.cs.uni-sb.de/tr/FB14`.
28. S. Diehl, editor. *Software Visualization*, volume 2269 of *LNCS State-of-the-art Survey*. Springer Verlag, 2002.
29. S. Diehl and A. Kerren. Levels of Exploration. In *Proceedings of the 32nd Technical Symposium on Computer Science Education, SIGCSE 2001*, pages 60–64. ACM, 2001.
30. S. Diehl, A. Kerren, and T. Weller. Visual Exploration of Generation Algorithms for Finite Automata. In *Implementation and Application of Automata, LNCS 2088*, pages 327–328, 2001.
31. Stephan Diehl, Carsten Görg, and Andreas Kerren. Animating Algorithms Live and Post Mortem. In *Proceedings of Dagstuhl Seminar on Software Visualization, 2001.*
32. R. A. Duisberg. Visual Programming of Program Visualizations. A Gestural Interface for Animating Algorithms. In *Proceedings of the 1987 IEEE Computer Society Workshop on Visual Languages*, pages 55–66, Linkoping, Sweden, August 1987.

33. P. Eades and K. Zhang, editors. *Software Visualization*. World Scientific Pub., Singapore, 1996.
34. M. Eisenstadt and M. Brayshaw. The Transparent Prolog Machine (TPM): An Execution Model and Graphical Debugger for Logic Programming. *Journal of Logic Programming*, 5(4):1–66, 1988.
35. S. Foubister. *Graphical Application and Visualisation of Lazy Functional Computation*. PhD thesis, Department of Computer Science, University of York, 1995.
36. Jaroslaw Francik. Algorithm Animation Using Data Flow Tracing. In *Proceedings of Dagstuhl Seminar on Software Visualization, 2001*.
37. P. A. Gloor. AACE - Algorithm Animation for Computer Science Education. In *Proceedings of the 1992 IEEE Workshop on Visual Languages*, pages 25–31, Seattle, WA, September 1992.
38. P. A. Gloor. Animated Algorithms. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 27, pages 409–416. MIT Press, Cambridge, MA, 1998.
39. P. A. Gloor. User Interface Issues for Algorithm Animation. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 11, pages 145–152. MIT Press, Cambridge, MA, 1998.
40. E. Gramond and S. H. Rodger. Using JFLAP to Interact with Theorems in Automata Theory. *SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education)*, 31, 1999.
41. J. Haajanen et al. Animation of User Algorithms on the Web. In *Proceedings of the 1997 IEEE Symposium on Visual Languages*, pages 360–367, Capri, Italy, September 1997.
42. E. Helttula, A. Hyrskykari, and K.-J. Räihä. Graphical Specification of Algorithm Animations with Aladdin. In *Proceedings of the 22nd Hawaii International Conference on System Sciences*, pages 892–901, Kailua-Kona, HI, January 1989.
43. R. R. Henry, K. M. Whaley, and B. Forstall. The University of Washington Illustrating Compiler. *Sigplan Notices: SIGPLAN '90*, 25(6):223–233, June 1990.
44. F. Hopgood. Computer Animation Used as a Tool in Teaching Computer Science. In *Proceedings IFIP Congress*, 1974.
45. A. Hyrskykari and K.-J. Räihä. Animation of Algorithms Without Programming. In *Proceedings of the 1987 IEEE Computer Society Workshop on Visual Languages*, pages 40–54, Linkoping, Sweden, August 1987.
46. S. Khuri and K. Holzapfel. 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, 2001.
47. K. Knowlton. L6: Bell Telephone Laboratories Low-Level Linked List Language. 16-minute black-and-white film, 1966.
48. K. Koskimies and K. Mössenböck. Scene: Using Scenario Diagrams and Active Text for Illustrating Object-Oriented Programs. In *Proceedings of the 18th IEEE International Conference on Software Engineering*, pages 366–375. IEEE Computer Society Press, 1996.
49. E. Kraemer. Visualizing Concurrent Programs. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 17, pages 237–256. MIT Press, Cambridge, MA, 1998.

50. E. Kraemer and J. T. Stasko. Toward Flexible Control of the Temporal Mapping from Concurrent Program Events to Animations. In *Proceedings of the 8th International Parallel Processing Symposium (IPPS '94)*, pages 902–908, Cancun, Mexico, April 1994.
51. S. K. Lodha, J. Beahan, T. Heppe, A. Joseph, and B. Zane-Ulman. MUSE: A Musical Data Sonification Toolkit. In *Proceedings of International Conference on Auditory Display (ICAD)*, Palo Alto, CA, USA, 1997.
52. R. L. London and R. A. Duisberg. Animating Programs Using Smalltalk. *Computer*, 18(8):61–71, August 1985.
53. Y. Moses, Z. Polunsky, and A. Tal. Algorithm Visualization For Distributed Environments. In *Proceedings of the IEEE Symposium on Information Visualization 1998*, pages 71–78, 1998.
54. S. Mukherjea and J. 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.
55. M. A. Najork. Web-Based Algorithm Animation. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pages 506–511, 2001.
56. T. L. Naps. Algorithm Visualization in Computer Science Laboratories. In *Proceedings of the 21st SIGCSE Technical Symposium on Computer Science Education*, pages 105–110, Washington, DC, February 1990.
57. T. L. Naps, J. R. Eagan, and L. L. Norton. JHAVE – An Environment to Actively Engage Students in Web-Based Algorithm Visualizations. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, pages 109–113, Austin, TX, March 2000.
58. James Noble. Visualising Objects: Abstraction, Encapsulation, Aliasing and Ownership. In *Proceedings of Dagstuhl Seminar on Software Visualization, 2001.*
59. M. Oudshoorn, H. Widjaja, and S. Ellershaw. Aspects and Taxonomy of Program Visualisation. In P. Eades and K. Zhang, editors, *Software Visualisation.* World Scientific Press, Singapore, 1996.
60. C. Pape. *Animation of Structured Proofs in Education at University Level (in German).* PhD thesis, University of Karlsruhe, Germany, 1999.
61. C. Pape and P. H. Schmitt. Visualizations for Proof Presentation in Theoretical Computer Science Education. In Z. Halim, Th. Ottmann, and Z. Razak, editors, *Proceedings of International Conference on Computers in Education*, pages 229–236. AACE - Association for the Advancement of Computing in Education, 1997.
62. W. C. Pierson and S. H. Rodger. Web-based Animation of Data Structures using JAWAA. In *Proceedings of the 29th ACM Technical Symposium on Computer Science Education (SIGCSE 98)*, pages 257–260, Atlanta, GA, February 1998.
63. B. A. Price, R. Baecker, and I. Small. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.
64. S. Robbins. The JOTSA Animation Environment. In *Proceedings of the 31st Hawaii Int. Conference on Systems Science*, pages 655–664, 1998.
65. G.-C. Roman. Declarative Visualization. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 13, pages 173–186. MIT Press, Cambridge, MA, 1998.
66. G.-C. Roman and K. C. Cox. A Declarative Approach to Visualizing Concurrent Computations. *Computer*, 22(10):25–36, October 1989.
67. G.-C. Roman, K. 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, June 1992.

68. G. Rössling and B. Freisleben. ANIMAL: A System for Supporting Multiple Roles in Algorithm Animation. *Journal of Visual Languages and Computing*, 2002. to appear.
69. G. Rössling, M. Schuler, and B. Freisleben. The ANIMAL Algorithm Animation Tool. In *Proceedings of the ITiCSE 2000 Conference*, pages 37–40, Helsinki, Finland, 2000.
70. H. Senay and S. G. Lazzeri. Graphical Representation of Logic Programs and Their Behavior. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, pages 25–31, Kobe, Japan, October 1991.
71. J. T. Stasko. Using Direct Manipulation to Build Algorithm Animations by Demonstration. In *Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems*, New Orleans, LA, USA.
72. J. T. Stasko. TANGO: A Framework and System for Algorithm Animation. *Computer*, 23(9):27–39, 1990.
73. J. T. Stasko. The Path-Transition Paradigm: A Practical Methodology for Adding Animation to Program Interfaces. *Journal of Visual Languages and Computing*, 1(3):213–236, 1990.
74. J. T. Stasko. The PARADE Environment for Visualizing Parallel Program Executions: A Progress Report. Technical Report GIT-GVU-95-03, 1995.
75. J. T. Stasko. Using Student-Built Algorithm Animations as Learning Aids. In *Proceedings of the 1998 ACM SIGCSE Conference*, San Jose, CA, 1997.
76. J. T. Stasko. Building Software Visualizations through Direct Manipulation and Demonstration. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 14, pages 187–203. MIT Press, Cambridge, MA, 1998.
77. J. T. Stasko. Smooth Continuous Animation for Portraying Algorithms and Processes. In John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors, *Software Visualization: Programming as a Multimedia Experience*, chapter 8, pages 103–118. MIT Press, Cambridge, MA, 1998.
78. J. T. Stasko, J. Domingue, M. H. Brown, and B. A. Price. *Software Visualization*. MIT Press, 1998.
79. J. T. Stasko and E. Kraemer. A Methodology for Building Application-Specific Visualizations of Parallel Programs. *Journal of Parallel and Distributed Computing*, 18(2), 1993.
80. J. T. Stasko and D. S. McCrickard. Real Clock Time Animation Support for Developing Software Visualisations. *Australian Computer Journal*, 27(4):118–128, 1995.
81. J. T. Stasko and J. F. Wehrli. Three-Dimensional Computation Visualization. In *Proceedings of the 1993 IEEE Symposium on Visual Languages*, pages 100–107, Bergen, Norway, August 1993.
82. E. Sutinen, editor. *Proceedings of the First Program Visualization Workshop 2000*, Porvoo, Finland, 2000. Department of Computer Sience, University of Joensuu, Finland.
83. A. Y. Tal and D. P. Dobkin. Visualization of Geometric Algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 1(2), 1995.
84. Ayellet Tal. Algorithm Animation Systems for Constrained Domains. In *Proceedings of Dagstuhl Seminar on Software Visualization, 2001*.
85. P. Vickers and J. L. Alty. CAITLIN: A Musical Program Auralisation Tool to Assist Novice Programmers with Debugging. In *Proceedings of International Conference on Auditory Display (ICAD)*, Palo Alto, CA, USA, 1996.

86. P. Vickers and J. L. Alty. Musical Program Auralisation: Empirical Studies. In *Proceedings of International Conference on Auditory Display (ICAD)*, Atlanta, GA, USA, 2000.
87. R. Watson and E. Salzman. A Trace Browser for a Lazy Functional Language. In *Proceedings of the Twentieth Australian Computer Science Conference*, pages 356–363, 1997.
88. J. R. Weinstein and P. R. Cook. FAUST: A Framework for Algorithm Understanding and Sonification Testing. In *Proceedings of International Conference on Auditory Display (ICAD)*, Palo Alto, CA, USA, 1997.