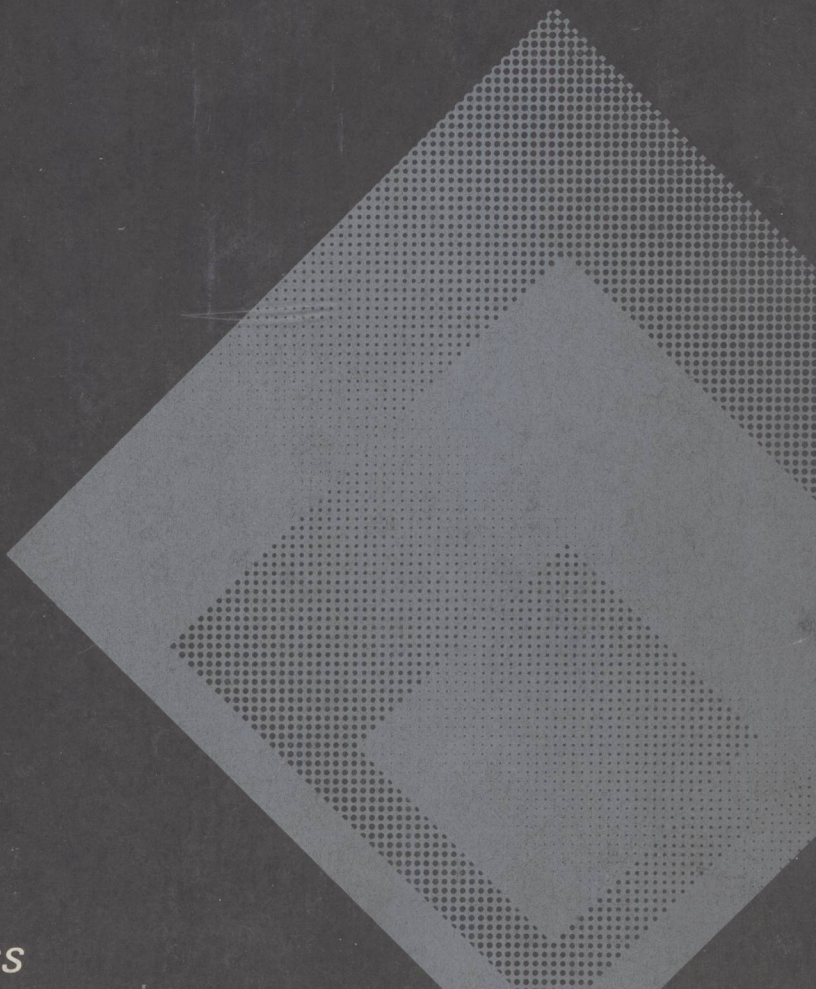
An ACM Distinguished Dissertation
1987

Algorithm Animation

Marc H. Brown



The MIT Press

Algorithm Animation

Marc H. Brown

The MIT Press
Cambridge, Massachusetts
London, England

Publisher's Note

This format is intended to reduce the cost of publishing certain works in book form and to shorten the gap between editorial preparation and final publication. Detailed editing and composition have been avoided by photographing the text of this book directly from the author's prepared copy.

Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Brown, Marc H.
Algorithm animation.

(ACM distinguished dissertations ; 1987)
Bibliograpy: p.
Includes index.
1. Electronic digital computers—Programming.
2. Algorithms. 3. Computer graphics. I. Title.
II. Series.
QA76.6.B765 1988 006.6 88-10036
ISBN 0-262-02278-8

To Ellen:

My best-friend and lover,
mi dushi and wife.

אֵשֶׁת חַיִל מִי יִמְצָא
וְרָחֹק מִפְּנִינִים מִכְרָהּ.

And to our parents

Preface

An algorithm animation environment is a means for exploring the dynamic behavior of algorithms that makes possible a fundamental improvement in the way we understand and think about them. It presents multiple graphical views of an algorithm in action, exposing properties that might otherwise be difficult to understand or even remain unnoticed. All views are updated simultaneously in real time as the algorithm executes, and each view is displayed in a separate window on the screen, whose location, size, and level of detail is interactively set by the end-user. A specialized interpreter controls execution in units that are specific to the algorithm, and allows multiple algorithms to be run simultaneously for comparisons. Programmers can implement new algorithms, graphical displays, and input generators and run them with existing libraries of such components.

An important aspect of an algorithm animation environment, and one we believe is useful in any interactive environment, is the concept of a *script*: a record of an end-user's session that can be replayed. Scripts can be used both passively, as virtual videotapes, and actively, as an innovative communication medium: the viewer of a script can customize the videotape interactively, and can readily switch between passively viewing it and actively exploring its contents. Scripts are typically used as high-level macros, system tutors, and electronic textbooks and research notes. End-users can create scripts easily by instructing the system to "watch what I do;" scripts are stored as readable and editable PASCAL programs.

The potential of such algorithm animation environments is great, but can be fully realized only if they are sufficiently easy and enjoyable to use. This dissertation is step towards achieving these goals. In it, we develop a model for creating real-time animations, as well as a framework for interacting with these animations. We also describe a prototype system, BALSA–II, and its feasibility study system, BALSA–I, that realize the conceptual model.

Acknowledgements

I've looked forward to writing this page for quite some time, having spent over a decade at Brown—as an undergraduate, TA, RA, staff member, and most recently, at the bottom of the totem pole, as a graduate student. My mentors merit more than the "thanks" that an acknowledgement page provides. Yet what does one say?

Bob Sedgewick has gone beyond a mere thesis advisor, professor or colleague. He has provided lodging and repasts—from Marly-le-Roi to Quonochontaug—7am squash matches, incisive interpretations of intra and interdepartmental matters, and a plethora of stimulating intellectual challenges. Sedge sets high standards for himself and those around him, and, more than anyone I know, appreciates one's obsessions for detail and perfection.

My competence as a programmer can be attributed, in part, to fundamental concepts drilled in Andy van Dam's freshman introductory programming course. Throughout the years, Andy's indefatigability, acumen, and vigorous leadership have inspired me and many other students to set high personal goals and to go after them. Computer science needs more people with Andy's genuine love for teaching and undaunted confidence in undergraduates.

The third member of my thesis committee, Paris Kanellakis, epitomizes the word "mensch." His encouragement, keen insight into technical matters, and careful reading of the final draft have been invaluable.

Many thanks to the people who read parts of this thesis for their perceptive comments. In particular, the prose has been greatly improved by two meticulous passes by Trina Avery, and by Cynthia Hibbard's patience and perseverance in explaining to me many intricacies of the English language. Special thanks to DAW for reacquainting me with [63] and for many inspired harangues and motivating discussions. Rob Rubin provided an exceptional sounding board, and frequently painted the global picture for me when I became myopic.

This thesis is an outgrowth of the Electronic Classroom project at Brown, initiated by Bob Sedgewick. The project realizes his vision of bringing "interactive movies" into computer science education and the study of algorithm. Bob, along with Andy van Dam and Tom Doeppner, procured the project's initial funding from NSF's CAUSE program in June, 1980. The Exxon Education Foundation and Apollo Computer provided additional funding.

Literally thousands of computer science students at Brown have taken courses in the Electronic Classroom using the BALSA–I environment. Many people, too numerous to mention here, have contributed to that software endeavor. In particular, Steve Reiss, with some help from Joe Pato and myself, developed the Brown Workstation Environment and tailored it to BALSA–I's requirements. Perry Busalacchi, Ham Lord, Karen Smith, Kate Smith, and Liz Waymire wrote many of the initial BALSA–I animations at a time when the system was unstable, under development, and without documentation. Mike Strickman implemented much of the BALSA–I kernel, Dave Nanian implemented its tools for broadcasting, and Eric Wolf implemented its code view. The hardware and system software was adeptly managed by Dave Durfee, Jeff Coady, and Dorinda Moulton. Their professionalism kept me (and others) in check during many hectic times.

Special credit is due to Bob Sedgewick for his many contributions to this thesis. While I can take full credit for the BALSA–I and BALSA–II systems design and implementation (subject to the acknowledgements cited above), neither system would have reached fruition without Bob's support and feedback. Jointly, we created the first sophisticated set of views and scripts using BALSA–I. Our experiences were instrumental in incremental improvements to BALSA–I and the entire design of BALSA–II. Appendix A discusses this experience and also Andy's experiences as instructor of an Introductory Programming course that used the Electronic Classroom at the same time.

Thus, it is important for readers to understand that when I use the pronoun "we" in this dissertation, it is more than the proverbial "royal we."

Most of the research reported here was performed while I was supported by an IBM Graduate Student Fellowship. My tenure as a graduate student was also supported in part by the ONR and DARPA under Contract N00014–83–K–0146 and ARPA Order No. 4786, and NSF Grant SER80–04974. Their assistance is gratefully acknowledged.

Finally, I am very thankful to Bob Taylor at DEC's Systems Research Center for providing support during the final months. His managerial style and motivation techniques (read: prohibiting me from participating in SRC projects until my dissertation is signed, sealed, and delivered) are creative. I hope my contributions to the SRC environment in the coming years will justify the confidence he has demonstrated.

Contents

1

Introduction

An algorithm animation environment is an "exploratorium" for investigating the dynamic behavior of programs, one that makes possible a fundamental improvement in the way we understand and think about them. It presents multiple graphical displays of an algorithm in action, exposing properties of the program that might otherwise be difficult to understand or might even remain unnoticed. All views of the algorithm are updated simultaneously in real time as the program executes; each view is displayed in a separate window on the screen, whose location and size is controlled by the end-user. The end-user can zoom into the graphical (currently, two-dimensional) image to see more detailed information, and can scroll the image horizontally and vertically. Views can also be used for specifying input to programs graphically. A specialized interpreter controls execution in units that are meaningful for the program, and allows multiple algorithms to be run simultaneously for comparing and contrasting. The end-user can control how the algorithms are synchronized by manipulating the amount of time each unit takes to execute. Programmers can implement new algorithms, graphical displays, and input generators and run them with existing libraries of algorithms, displays, and inputs.

While an algorithm animation environment is a rich environment for actively exploring algorithms, in many situations a passive, guided approach using a prepared "script" is more appropriate. For example, when dynamic material is a visual aid in a lecture or when it complements a traditional textbook or journal article, the audience is interested in viewing the "virtual videotape" as the author conceived it, not in exploring the material independently. And when an algorithm is being viewed for the first time, self-guided exploration can easily result in distorted or incorrect interpretations and leave important aspects of the algorithm undiscovered.

To a first approximation, an algorithm animation script is a record of an end-user's session that can be replayed. At one end of the spectrum, a script is merely a videotape that can be viewed passively. At the other, more interesting end of the spectrum, a script is an innovative communication medium: the viewer of a script can customize the movie interactively, and can readily switch between passively viewing it and actively exploring its contents. Scripts can be used as high-level macros, thereby extending the set of commands available to end-users of the algorithm animation system, and can also serve as the basis for broadcasting one end-user's session to other end-users on other machines. End-users can create scripts easily by instructing the system to "watch what I do." Scripts are stored as readable and editable PASCAL programs.

Systems for algorithm animation can be realized with current hardware: personal workstations—with their high-resolution displays, powerful dedicated processors, and large amounts of real and virtual memory—can support the required interactiveness and dynamic graphics. In the future, such workstations will become cheaper, faster, and more powerful, and will have better resolution. An algorithm animation environment exploits these characteristics, and can also take advantage of a number of features expected to become common in future hardware, such as color, sound, and parallel processors.

We develop here a model for creating real-time animations of algorithms with minimal intrusions into the algorithm's original source code, as well as a framework for interacting with these animations. We also describe a prototype system, BALSA–II, and its feasibility study system, BALSA–I, that realize the conceptual model. Currently, BALSA–II is being used in teaching parts of a data structures course, for research in the design and analysis of algorithms, and for technical drawings in research papers and textbooks. BALSA–I has been in production use since 1983 in Brown University's "Electronic Classroom." Appendix A documents some of our experiences using the environment as a principal mode of communication during lectures in an introductory programming course and in an algorithms and data structures course. Appendix B cites publications describing various aspects of the project.

1.1 Thesis Contributions

The primary contributions of this thesis are its models for (1) programmers creating animations, (2) end-users interacting with the animations, and (3) end-users creating, editing, and replaying dynamic documents. These models have been realized in the BALSA–I and BALSA–II systems. A secondary contribution of this research is the numerous static and dynamic graphical displays of a wide range of algorithms and data structures we have created using the prototype systems, most of which had never been displayed or even conceived previously. The domain includes sorting, searching, string processing, parsing, graphs, trees, computational geometry, mathematics, linear and dynamic programming, systolic architectures, and graphics. The systems have also be used to show innovative dynamic illustrations of fundamental concepts in procedural programming languages. The diagrams in this document are a small sampling of these images; others have been reported elsewhere [16, 17, 18].

We now elaborate on the primary contributions of this thesis.

(1) **A model for programmers creating animations.** The programmer model is independent of the contents of all algorithms, inputs and views; hence, it can be used to animate any algorithm in a systematic manner. Moreover, the model makes it easy to animate new algorithms and create new displays.

Algorithms being animated are separated into three components: the *algorithm* itself, an *input generator* that provides data for the algorithm to manipulate, and graphical *views* that present the animated pictures of the algorithm in execution. Views are built following a classical graphics *modeler–renderer* paradigm, and an *adapter* allows any particular view to be used to display aspects of many different algorithms. Modelers can be chained to provide views of views, and renderers can be based on multiple modelers. Algorithms are annotated with *interesting events* to indicate phenomena of interest that should give rise to the displays being updated; in addition, the events provide the abstraction for end-users to control the execution.

(2) **A model for end-users interacting with animated algorithms.** The end-user model, like the programmer model, is independent of the algorithms, inputs, and views; thus, end-users interact with animations

in a consistent manner. This model gives well-defined semantics for each end-user command; in fact, BALSA–I and BALSA–II can be thought of as merely two different user interfaces that manipulate these properties.

The interactive environment is characterized at any point by its *structural*, *temporal* and *presentation* properties. The structural properties are the set of algorithms currently running and the data they are processing. Information concerning the specialized interpreter, such as the program-specific units chosen for stopping and stepping points and how multiple algorithms are synchronized, are the temporal properties. The configuration of view windows on the screen are considered the presentation properties. In addition to providing data for algorithms to process, end-users can manipulate the underlying algorithms, input generators and views through the concept of *parameters* for each component. That is, end-users can select among parameters preset by the programmers; end-users cannot create new variants at runtime.

(3) **Model for end-users creating, editing, and replaying dynamic documents.** Dynamic documents, called *scripts*, are created by having the system watch what the end-user does. However, a semantic interpretation of the actions is maintained in a textual file—an executable PASCAL program—not a command or keystroke history. Scripts form a basis for passively watching the dynamic material like a videotape, or actively interacting with the material. The script model is mostly independent of an algorithm animation system: the principals can be applied to virtually any system with well-defined structural, temporal and presentation properties.

1.2 Applications of Algorithm Animation

An obvious application of an algorithm animation environment is computer science *instruction*, particularly courses dealing with algorithms and data structures, e.g., compilers, graphics, databases, algorithms, programming. Rather than using a chalkboard or viewgraph to show static diagrams, instructors can present simulations of algorithms and programming concepts on workstations. Moreover, students can try out the programs on their own data, at their own pace, and with different displays (from a library of existing

displays) from those the instructor chose. Non-naive students can code their own algorithms and utilize the same set of displays used by the instructor in demonstration programs. As mentioned earlier, the appendices describe how BALSA–I was used in computer science instruction.

Another proven application of an algorithm animation environment is as a tool for *research* in algorithm design and analysis. Human beings' ability to quickly process large amounts of visual information is well documented, and animated displays of algorithms provide intricate details in a format that allows us to exploit our visual capabilities. For instance, experimenting with an animation of Knuth's dynamic Huffman trees [43] revealed strange behavior of the tree dynamics with a particular set of input. This lead to a new, improved algorithm for dynamic Huffman trees [70]. A variation of Shellsort was discovered in conjunction with static color displays of Bubblesort, Cocktail-Shaker Sort, and standard Shellsort [41]. An early version of BALSA–I was used to help understand and analyze a newly discovered stable Mergesort [36].

Animations developed for instruction can be used for research, and vice versa. For example, animations for an algorithms course were used with minor changes to investigate shortest-path algorithms in Euclidean graphs [61]. Conversely, displays developed in conjunction with research on pairing heaps [29] were later incorporated into classroom lectures on priority queues.

Another application for an algorithm animation environment is as a testbed for *technical drawings* of data structures. It allows interactive experimentation with input data and algorithm parameters to produce a picture that best illustrates the desired properties. Furthermore, the drawings produced are always accurate, even ones which would tax the best of draftsmen. For example, it is laborious indeed for a draftsman to take a set of points and prepare the sequence in Figure 1.1 showing the construction of a Voronoi diagram and its dual. As more and more researchers begin to typeset their own papers and books, this application will become increasingly important.

A prime but so far unexplored application area for algorithm animation is in *programming environments*. Pioneering environments on graphics-based workstations, such as Cedar [65], Interlisp [64], and especially Smalltalk [31], are, by and large, text-oriented. Recent workstation-based program development environments incorporate graphical views of the program structure and code, not data, and consequently have had limited success in giving additional insight into the programs: "The experience we have had with

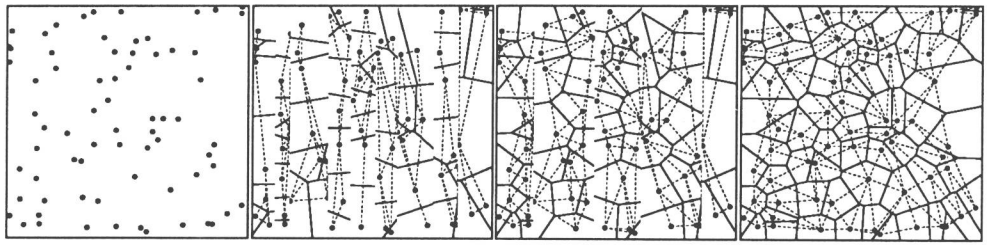


Figure 1.1: *Construction of a Voronoi diagram.*

PECAN, however, has shown that such graphical views are limited in their power and usefulness when they are tied to syntax. The syntactic basis forces the user to treat these two-dimensional representations in a one-dimensional way, and the graphics do not provide any significant advantage over text" [57]. These systems could be greatly enhanced by the display capabilities of an algorithm animation environment.

Algorithm animation has also been used for *performance tuning* [24], and has the potential to be helpful in *documenting programs* [47] and in *systems modeling*, especially for multi-threaded applications.

1.3 Conceptual Model

Algorithm animation involves two types of users: *end-users* and *client-programmers*. The end-users watch and interact with the animations on a workstation, whereas the programmers implement the algorithms, displays, and input generators that the end-users see and manipulate. An algorithm animation system itself is domain-independent: the system does not know whether an algorithm sorts numbers or produces random numbers, or whether a view shows a tree or a table. It does not attempt to decide what phenomena are interesting in a program, or what styles of input or visual representations are appropriate. Rather, it provides tools so that a large variety can be easily implemented and end-users can watch and interact with them in a consistent manner.

We will use the terms *algorithm animation environment* and *algorithm animation system* to reflect the two types of users. The algorithm animation

system is the code with which client-programmers interface, and the algorithm animation environment is the runtime environment that end-users see. It is the result of compiling the code that client-programmers implement with the algorithm animation system.

For end-users, the main goal of the algorithm animation environment is to provide a consistent manner in which to interact with animations, independent of who happened to prepare the animation and what domain the animation happens to be from. Once an end-user has used the system for one algorithm, he should know how to use it for any and all algorithms.

For client-programmers, the main goal of an algorithm animation system is to provide all of the ancillary functions needed to make an interactive animation. Each programmer should not need to reinvent and reimplement facilities common to all views, such as zooming into displays. A second important goal is to provide a model whereby the animation code is separated from the algorithm. Moreover, the code relating to the animation (and to preparing input for an algorithm) should be shareable by many algorithms. Thus, a programmer implementing an algorithm should be able to concentrate primarily on the algorithm, independent of input generators and displays and the window configuration selected by the end-user. Conversely, a programmer implementing a display should do so without concern for the algorithm, input generators or the end-users.

The program being animated must be split into various pieces so that the algorithm animation system, as well as the end-users, can manipulate them systematically. Programs are separated into three components: the *algorithm* itself, the various *input generators* that provide data for the algorithm to manipulate, and the various *graphical displays*, or *views*, that present the animated pictures of the algorithm in execution.

The remainder of this section presents a high-level overview of the model an algorithm animation systems gives to its two types of users. The descriptions of the models here are necessarily brief and incomplete; Chapters 3 and 4 are devoted to end-users, and Chapters 5 and 6 to programmers.

End-User's Model

The end-user of an algorithm animation environment is always in a "setup-run" loop:

Setup: The end-user arranges the screen and decides which algorithms to

run, which input generator and views to use, and what the values of any parameters to each of these should be. Each algorithm has a default setup that can be designated and changed by the end-user.

Run: The end-user runs the algorithms and watches them in the view windows on the screen. While the algorithms are running, the end-user can suspend them to change the ensemble of views on the screen as well as the program's speed and breakpoints.

Changing or creating the content of an algorithm, view, and input generator is done, strictly speaking, not by an end-user but by a programmer. Such editing is done outside of the algorithm animation environment, using the standard editors and compilers. If the machine supports multiple processes as well as dynamic loading and unloading, the algorithm animation environment does not have to be exited.

Because the notion of parameters to algorithms, input generators, and views is rather unconventional, we now elaborate.

Algorithms, input generators, and views can all be tuned directly by the end-user. Just what the parameters mean for any particular algorithm, input generator, or view depends on how it was implemented. Thus, it is the programmer, not the end-user, who decides what the parameters are—whether the particular component will even have any parameters, what the user interface will be that controls how they are set, what their default values are, and so forth. The user interface management tools and guidelines of the underlying workstation environment promote consistency in the user interface for manipulating parameters across many domains.

Algorithm parameters affect the algorithm, not the data that the algorithm manipulates. For example, should the lexical analyzer in an animated compiler use a hash table of size 119 or 2001? Or should it use a binary tree (or some specific type of balanced tree) rather than a hash table? *Input parameters* affect the input generators. For example, the input parameter to a generator that reads numbers from a file for sorting algorithms would be the name of the file. Another generator for the same sorting algorithm might produce random numbers; the parameters for this generator would be how many numbers to produce and a seed for a random number generator. Of course, an input parameter will affect the data indirectly, which in turn will affect the algorithm. *View parameters* affect how information is displayed in a particular view window; they do not affect the algorithm