98

The 1992 SRC Algorithm Animation Festival

Marc H. Brown

March 27, 1993

Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center (SRC) and three other research laboratories are committed to filling that need.

SRC began recruiting its first research scientists in l984—their charter, to advance the state of knowledge in all aspects of computer systems research. Our current work includes exploring high-performance personal computing, distributed computing, programming environments, system modelling techniques, specification technology, and tightly-coupled multiprocessors.

Our approach to both hardware and software research is to create and use real systems so that we can investigate their properties fully. Complex systems cannot be evaluated solely in the abstract. Based on this belief, our strategy is to demonstrate the technical and practical feasibility of our ideas by building prototypes and using them as daily tools. The experience we gain is useful in the short term in enabling us to refine our designs, and invaluable in the long term in helping us to advance the state of knowledge about those systems. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

SRC also performs work of a more mathematical flavor which complements our systems research. Some of this work is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. The rest of this work explores new ground motivated by problems that arise in our systems research.

DEC has a strong commitment to communicating the results and experience gained through pursuing these activities. The Company values the improved understanding that comes with exposing and testing our ideas within the research community. SRC will therefore report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

The 1992 SRC Algorithm Animation Festival

Marc H. Brown

March 27, 1993

Abstract

During the last two weeks of July 1992, twenty researchers at Digital Equipment Corporation's Systems Research Center participated in the 1st Annual SRC Algorithm Animation Festival. Only two of the researchers had previously animated an algorithm, and not too many more had ever written an application that involved graphics. In this paper, we report on the Animation Festival, describing why we did it and what we did, and commenting on what we learned.

Preliminaries

By and large, computer scientists are dogmatic about the programming language they use and are loath to change. While this attitude is not necessarily bad (clearly there are real costs associated with learning about and changing to a new language), it is one of the many hurdles facing a new programming language, even one with many technical advances.

One way to promote a new programming language is to provide an extensible application with some “steak and sizzle” implemented in the particular language. That is, an application that is useful for some real tasks (the “steak”), that is fun and enticing to use (the “sizzle”), and that can be tailored to new tasks (the extensibility).

The 1992 SRC Algorithm Animation Festival, held during the last two weeks of July at Digital Equipment Corporation’s Systems Research Center (SRC), explored this approach to help promote the Modula-3 programming language within university computer science departments. Modula-3 [7] is a procedural language in the tradition of Pascal and Modula-2. It maintains the simplicity and safety of the earlier languages, while adding features necessary for developing large systems, such as objects, threads, exceptions, garbage collection, and a rich set of libraries. In this paper, we report on the Animation Festival, describing why we did it and what we did, and commenting on what we learned.

The primary goal of the Animation Festival was to promote Modula-3 by developing a comprehensive set of animations of fundamental algorithms to accompany a university-level data structures or algorithms course. However, the animations are not videotapes that one watches passively; rather, they are part of an exploratory computer environment in which students and teachers can study the algorithms. Furthermore, the algorithm animation environment, implemented in Modula-3, was designed so that it would be easy for non-experts to implement additional algorithm animations of their choice.

We did not expect professors to change their algorithms or data structures courses to use Modula-3. Rather, we simply wanted to make professors — especially those of undergraduate courses — aware of the language. Naturally we hoped that eventually the language would build a large following in universities (and elsewhere) based on its technical merits; but a necessary first step is to make professors aware the language.

We believed that it would be significantly easier for professors to incorporate animated algorithms into their classes if the visual representations in the animations corresponded closely to the static diagrams appearing in the textbook they were already using. Therefore, we chose to link our animations to Robert Sedgewick’s

Algorithms [8]. Not only does that text have a substantial number of course adoptions, but many of the diagrams in that text have their roots in algorithm animations [2].

We felt it was important for students to see snippets of code displayed as part of an animation in the same language used by the textbook, independent of the language used to implement the algorithm animations. Our choice of textbook presented us with an additional challenge because there are four versions of the text—Pascal, C, C++, and Modula-3. (The Modula-3 version was written after the Animation Festival.) We'll return to the issue of "code views" later.

The figures in this paper are screen dumps from animations produced by the participants during the Animation Festival. Of course, a static black-and-white image cannot do justice to an interactive color animation.

This paper is organized as follows: The next section describes the logistics of the Animation Festival. We then describe the software system infrastructure. Following that, we describe the current status of the software and compare it with other related efforts. And finally, we offer some concluding thoughts.

The Animation Festival

Preparation for the Animation Festival began in April '92 and continued through July. A team of six took on the challenge of building an algorithm animation system in which a non-expert could produce animations without ever having to become an expert. The six were to be the Animation Festival instructors. An additional twenty people participated as algorithm animators. We assumed that all participants were talented programmers, but without experience writing applications with graphical output or animating algorithms.

The Animation Festival lasted for two weeks; the first week was devoted to teaching the participants about algorithm animation, and the second week to implementing animations. The participants demonstrated their work to the rest of SRC during the following weeks.

The classes during the first week typically consisted of a couple of hours of lectures in the morning followed by some specific programming assignments. The lectures covered three general areas: using the algorithm animation system, creating effective animations, and programming in Modula-3.

During the second week, the participants formed groups and chose an algorithm or group of related algorithms from our "Most Wanted List" and animated them. Participants and instructors met at the end of each day for some R&R, that is "rushes and refreshments." This was an opportunity for participants to demonstrate what they had done that day and to get feedback from other participants and the instructors. Most groups had a basic animation up and running within a day of starting, and spent most of their time refining the animation.

About the participants

Of the twenty participants, two were graduate students who were at SRC as part of its Summer Research Internship program, one was a member of Digital's Paris Research Laboratory, and the others were fulltime members of the research staff at SRC. Their research interests were quite varied. Seven participants were primarily theoreticians (analysis of algorithms, algebraic geometry, combinatorics, etc.). Six participants were developers of interactive applications. The remaining seven participants were library and systems builders.

The participants formed 11 groups: one group of three, 7 pairs, and 3 singletons. The participants chose their own partners or chose to work alone. The benefits and drawbacks of team (programming) projects are well documented in the literature, and we certainly did not make any formal study. We observed the following phenomena:

- The animations developed by multi-person groups were noticeably better than those developed by single-person groups.
- All multi-person groups completed the animations they set out to develop within the two weeks. Of the three singletons, one dropped out and the other two finished a few weeks late.
- The members of multi-person groups were uniformly more enthusiastic about the experience than the people who worked alone.

Each multi-person group seemed to develop its own working style: In about half, the partners worked independently, sometimes sequentially (participants had travel and other constraints) and sometimes concurrently (in separate offices). In the other half, the partners did "team programming": the majority of the coding was done in one office by one of the partners, and the other partners looked on and offered suggestions and moral support.

Naturally the participants had their own motives for participating. The most common reasons were as follows:

- The Animation Festival provided participants an opportunity to work closely with other members of SRC with very different technical interests.

- The Animation Festival was an excellent opportunity for participants to learn Modula-3. (Only seven of the participants had previously written any programs in Modula-3.)

- Finally, animations of algorithms are nifty! The Animation Festival gave participants an opportunity to discover the magic of algorithm animation.

About the animations

As expected, the quality of the animations varied considerably. Three were considered by most viewers as outstanding and four others were on par with animations produced by people with significant experience such as the instructors. A number of observations are noteworthy.

An animation of Convex Hulls (Fig. 5) by Lyle Ramshaw and Jim Saxe started with the code from the textbook and used representations similar to those in the text. Soon, they developed a variation of the standard Package Wrap algorithm that elegantly handled degenerate inputs (e.g., collinear points). They cleverly named their program variables with the colors of the highlights that were used in the animation to depict their values. The choice of variable names provided a common reference point for relating the "code view" in the bottom-right with the graphical views in the top-left and top-right.

Steve Glassman and Greg Nelson animated Euclid's proof of the Pythagorean Theorem (Fig. 9). The view in the top-left contains the prose from Euclid's proof; as key phrases in the proof are encountered, they are highlighted and the figure in the graphical view at the top-right changes to reflect the proof. This animation suggests a new and exciting area to explore: animated proofs.

Finally, all of the Animation Festival participants used smooth transitions in their views rather than showing state changes with discrete transformations. Participants uniformly found smooth transformations more intriguing and engaging, both from programmer and pedagogical perspectives.

The Software System

This section describes Zeus, the software system we used in the festival for animating the algorithms. Our starting point was the Modula-2 implementation [1]. Porting Zeus to Modula-3 was straightforward; we then enhanced it in a variety of ways.

Readers familiar with Zeus will recall that the essence of animating an algorithm in Zeus is to separate the *algorithm* from each *view* and to create a narrow interface, specific to each algorithm, between them. More specifically, an algorithm (as one would find in a textbook) is annotated with procedure calls that identify the fundamental operations that are to be displayed. An annotation, called an *interesting event*, has parameters that identify program data. A view is a subclass of a window, with additional methods that are invoked by Zeus whenever an interesting event happens in the algorithm. Each view is responsible for updating its graphical display appropriately, based on the interesting events. Views can also propagate information from the user back to the algorithm.

We now describe Zeus from the user's perspective, concentrating on the new features that were added for the Animation Festival, and then outline the implications of these new features on programmers developing algorithm animations.

User Perspective

The user model in Zeus is simple: create a *scene* (or load it from a file) and then *run* the algorithm. A scene consists of specifying an algorithm to run, providing it with some input data, and specifying views in which to observe the algorithm. Each view is a separate top-level window. For example, Fig. 6 contains four views plus the Zeus control panel.

While the algorithm is running, the user can stop it and single-step through it. The user can also control the speed of the animation. Single-stepping is done in terms of the interesting events in the algorithm. The user can set which events constitute a "step"; that is, the algorithm will advance until the next event from this set is generated.

The most significant features that we added to the Modula-2 version of Zeus were *multiple algorithms*, *hand-simulation views*, and *scripting*.

Multiple algorithms

The user can specify any number of algorithms to run concurrently. Each algorithm is run in its own thread, and the threads are synchronized by Zeus after each event. That is, all algorithm threads are allowed to advance until the next event is generated. Then, each one waits until all algorithms have reached an event.

The figures in this paper contain two examples of multiple algorithms. Fig. 2 shows two search tree algorithms running together. The algorithm seen in the view at the top-left is a balanced tree algorithm using red-black binary trees whereas the algorithm in the lower-right is a standard binary search tree. Fig. 10 shows three parsing algorithms, each with the same two views. All three algorithms are processing the same input string.

Multiple algorithms are often used for algorithm races, that is, simply to observe the relative speeds of different algorithms. However, in the two examples just mentioned, the multiple algorithms are not being raced one against another to compare relative speeds. Rather, they are being run simultaneously to better understand the similarities and differences of how they operate.

In algorithm races, it is possible for a user to control the relative speeds of the algorithms by assigning a weight to each event for each algorithm. By default, the weight for all events is 1. Changing the weight of event e for algorithm a to 3, say, would mean that whenever algorithm a generated event e it would not proceed when all other algorithms also reached an event, rather, it would wait until all other algorithms generated 3 more events. This feature allows a user to simulate different models of execution. For example, one could run two instances of the same Convex Hull algorithm on the same input: in one instance, data movement might be the same cost as data access; in the other instance, data movement might be expensive and data access cheap.

Hand-simulation views

In a canonical algorithm animation, one sees purely graphical views of the algorithms, usually running on a fairly large amount of data. (Certainly more data than you'd be willing to hand-simulate the algorithm on!) However, it is also helpful to see the algorithm running on a small set of data, and to hand-simulate it, especially when first learning about the details of the algorithm. Zeus provides three types of views that are most applicable to situations where one would hand-simulate the algorithm: a code view, a data view, and an album view.

First, a *code view* highlights each line of code as it is executed. See the top two windows of Fig. 4 and the bottom-left window of Fig. 5. As we shall describe later, a code view highlights lines of text based on markers inserted into the algorithm; the actual text being highlighted is stored in an external file. In Fig. 4, the Zeus programmer took advantage of this feature to display multiple code views simultaneously: one in C and one in Modula-3. (Of course, this isn't possible to do for arbitrary programs; but for many algorithms, on the order of a page or so of code, the difference between the implementation languages is mostly syntactic sugar. The same set of markers can be used to drive different input text files.)

Fig. 9 shows another innovative use of the Zeus code view facilities. Here, the "code" is the text of the proof, as it appears in Euclid's *Elements*. Each paragraph of the proof is a "procedure," and a parameter to the code view causes it to display procedure calls by replacing the caller by the callee, rather than overlapping the caller by the callee, as in Fig. 4.

The second type of view for hand-simulating algorithms, a *data view*, displays the values of variables each time they change. Fig. 6, top-right, shows a data view of an integer array, and Fig. 7, top-right, shows a data view containing three integer variables.

Finally, when hand-simulating small amounts of data, it is often useful to see a history of the state of the views each time something interesting happens. Zeus provides an *album view* to do just this: at any point, the user can select “Photo” from a menu, and a miniature pixmap of the contents of all views will appear in the album view; see Fig. 9. (The album view is also useful when running an algorithm with large amounts of data, as seen in Fig. 2, where a photo was taken after every 10 nodes were added to the trees.)

Scripting

It is a challenging task to actually incorporate an animation into a classroom lecture. There are many content issues such as which examples to show, but there are also quite a few logistic issues such as remembering which views to create when, which data to run when, at what speed, and so on. One tool that has proven to be very helpful for dealing with the logistic issues is scripting. Put simply, a script is a high-level recording of a user’s session that can be replayed.

The scripting facility works in terms of *frames*, or runs of an algorithm. Each time the user starts running the algorithm, a textual description of the scene (i.e., algorithms, data, views, speed, and so on) is saved in the script file. At playback time, the scene is restored from this snapshot, and the algorithm starts executing.

The scripting facility ignores most user actions, such as those that suspend and restart algorithm execution. However, it is often useful to force a pause during playback. This can be done using the "Future Pause" button. This interrupts the algorithm’s execution during playback, and forces the user to click "Resume" to continue.

Programmer Perspective

The model given to the programmer creating an animation has not changed significantly from the Modula-2 implementation of Zeus. The addition of scripting, the album view, and multiple algorithms does not affect the programmer in any way; creating code views and data views, however, does require some work by the programmer.

There are two parts to a code view: First, the programmer annotates the algorithm with *abstract line numbers*. The annotation is just an interesting event,

Fig. 1: Pre-Festival Hacking

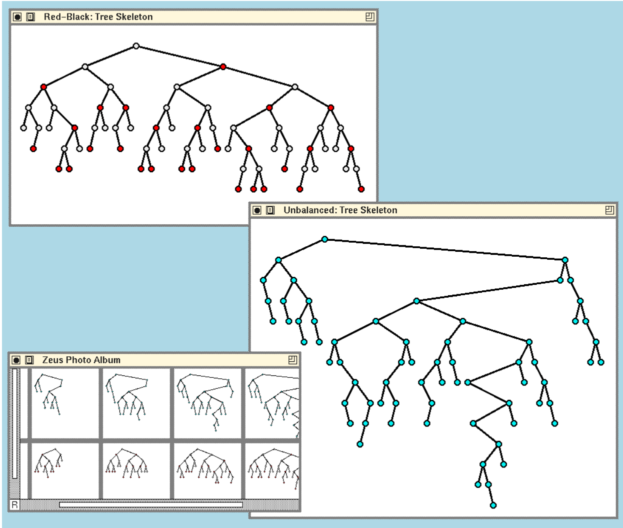

Fig. 2: Balanced Trees

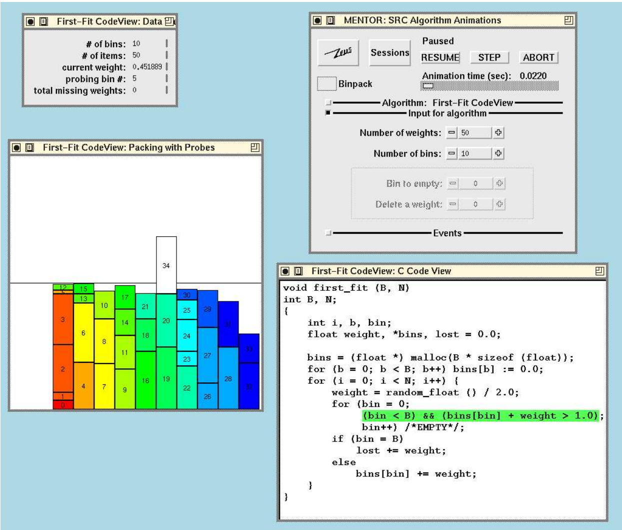

Fig. 3: Binpacking

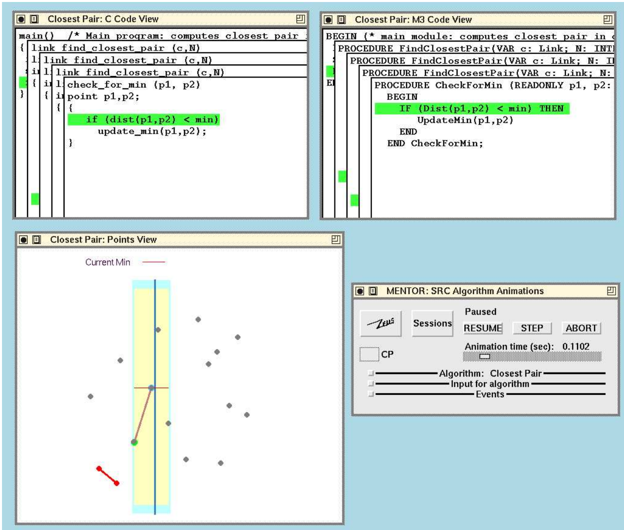

Fig. 4: Closest Point

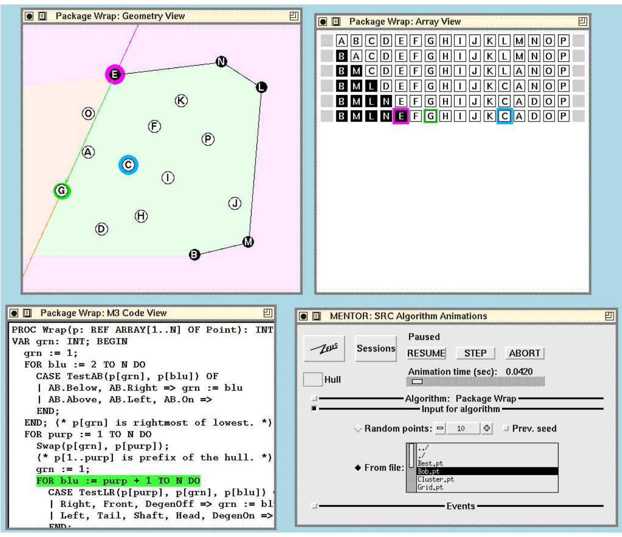

Fig. 5: Convex Hulls

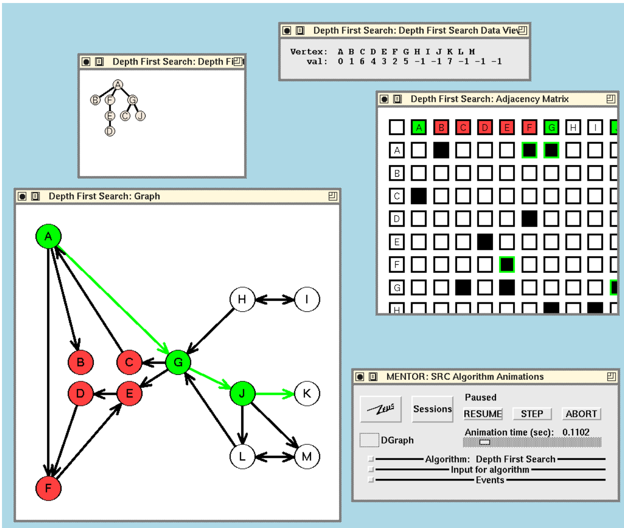

Fig. 6: Directed Graphs

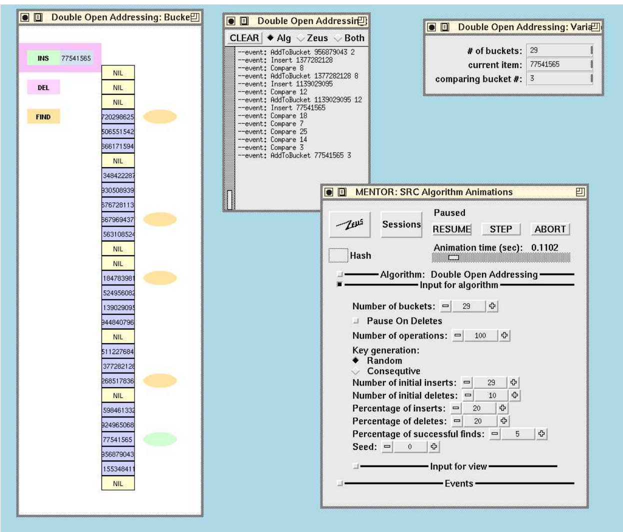

Fig. 7: Hashing

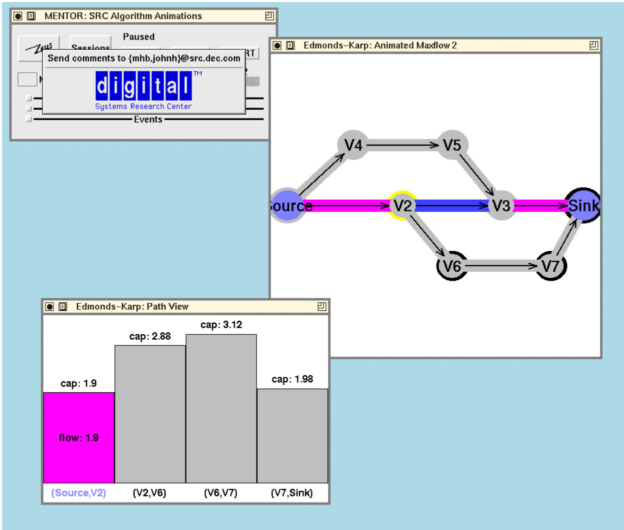

Fig. 8: Network Flow

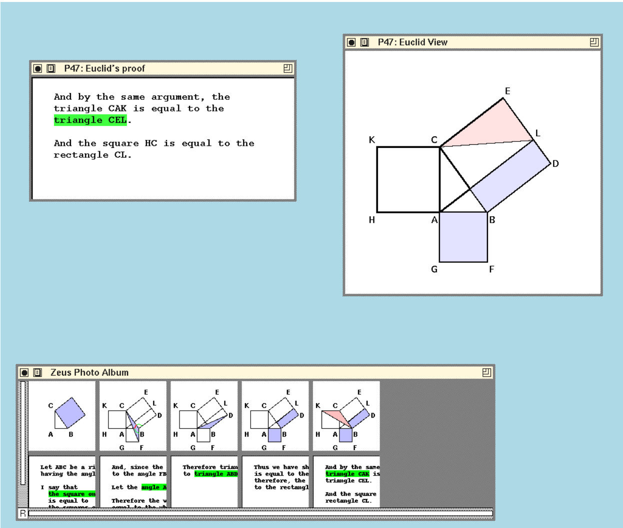

Fig. 9: Euclid's Proof

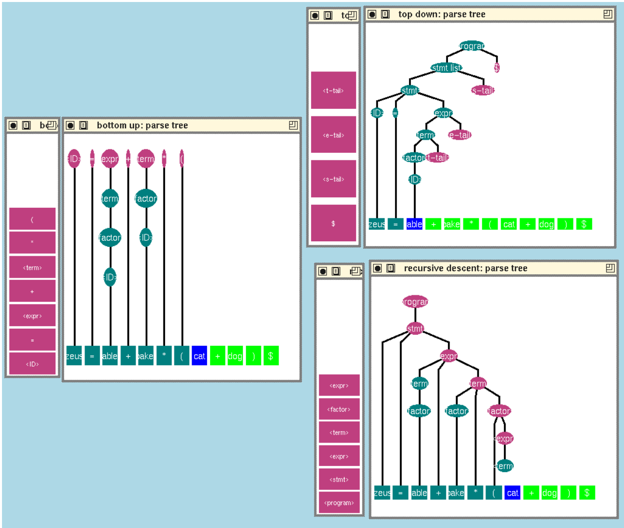

Fig. 10: Parsing

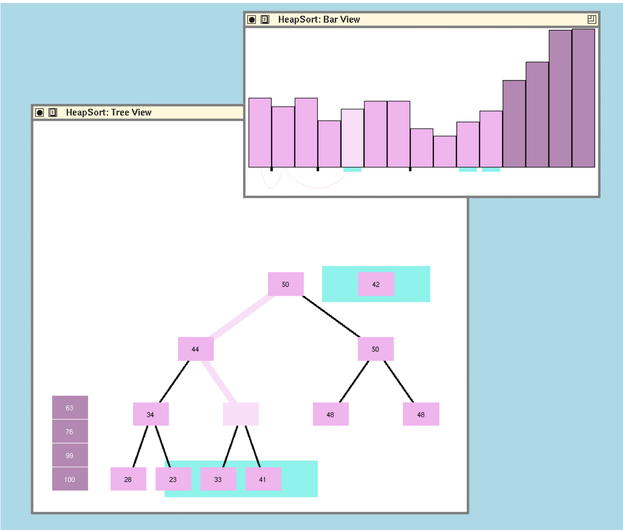

Fig. 11: Priority Queues

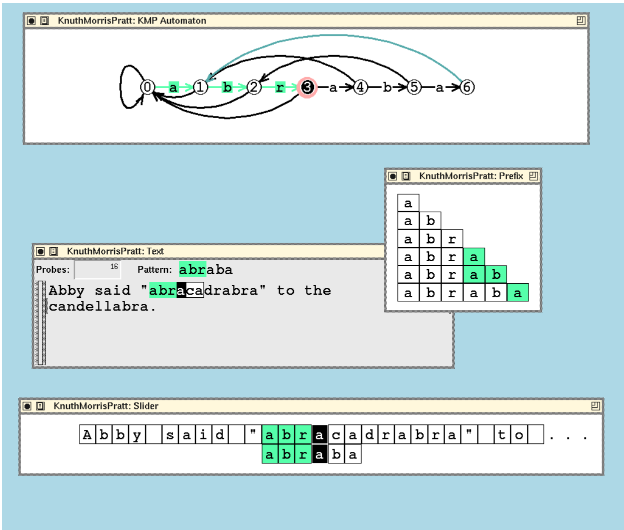

Fig. 12: String Searching

but the name of the event is predefined. The code view interesting event takes a single parameter, the abstract line number. Second, the programmer specifies a text file that will be displayed by each code view instance. Each time that an abstract line number is encountered when the algorithm executes, a section of text in each code view instance is highlighted. The contents of the text files are left up to the programmer, and Zeus provides an interactive tool for specifying which part of the text file corresponds to each abstract line number.

The heart of the data view implementation is support provided by the Modula-3 compiler for monitoring variables. When a variable is declared, a procedure can be registered (using a Modula-3 `TRACE` pragma) that will be invoked each time the value of the variable changes at runtime. The programmer must create a graphical user-interface for displaying the variables. This is typically done interactively using an application builder. (By default, the variables are displayed textually, but this code can be overridden to display variables in more interesting ways. However, none of the Animation Festival participants explored this option.)

The most important new tool provided for programmers is a 2-D animation package, GraphVBT [4], which is very easy to use. That package is described elsewhere, so we won't go into details here.

Related Work

The AACE project at MIT also plans to produce a set of animations of fundamental algorithms [6]. Their system consists of a hypertext version of an algorithms textbook (*Introduction to Algorithms* by Corman, Leiserson, and Rivest), with animations of many of the most common algorithms and some tools to analyze them. The system is implemented entirely in HyperTalk and runs in a standard HyperCard environment.

The AACE system is probably easier than Zeus for students to learn to use, primarily because it runs within the standard HyperCard environment and follows its user interface guidelines. However, because AACE is implemented as a HyperCard stack, it does not support multiple algorithms or multiple views, two important features available in Zeus.

We believe that learning to program a new animation is also easier in AACE than in Zeus, since there is no system *per se* to learn about, just the HyperTalk language. However, for an experienced programmer, creating sophisticated animations is probably easier in Zeus because of the rich animation libraries it has and because it is a framework designed specifically for animating algorithms.

Although the focus of this paper has not been on the Zeus system itself, the reader interested in algorithm animation systems should refer to BALSA [3] and TANGO [9], two systems that greatly influenced the design of Zeus, and to the earlier report on Zeus [1].

Conclusions

As a result of the Animation Festival, a significant amount of interesting and useful Modula-3 code was implemented. This includes animations of about a dozen families of algorithms, an algorithm animation system, and sophisticated graphics libraries. We are still working on our original goal of producing a comprehensive set of animations of fundamental algorithms.

The Animation Festival had a number of intangible results that are perhaps even more important: SRC researchers had an opportunity to work closely with colleagues with very different technical backgrounds and interests, and many of the participants developed new appreciations for the power of graphics and animation as communication mediums.

Although the graphics package provided to the participants of the Animation Festival was powerful and easy to use, the most cumbersome part of developing an animation was still implementing the views. Steve Glassman is currently developing a high-level, rapid-turnaround environment [5] based on Zeus and GraphVBT.

Finally, was the Animation Festival a success? The simple answer is to state that the *2nd Annual SRC Algorithm Animation Festival* is scheduled for the summer of 1993 and it appears to be fully subscribed! We expect some participants will explore both audio and 3-D graphics as tools for communicating the workings of programs, as they animate some of the many fundamental (and not so fundamental) algorithms that have not been animated within the Zeus system,

System Availability

The Zeus system and the animations developed during the Animation Festival are available via anonymous ftp from `gatekeeper.dec.com`. They are located in the directory `pub/DEC/Modula-3/release`.

You can find out information about Modula-3 in the Usenet news group `comp.lang.modula3`. If you do not have access to Usenet, you can be added to a relay mailing list by sending a message to `m3-request@src.dec.com`.

Acknowledgments

The instructors in the Animation Festival were Mike Sclafani, John Hershberger, Stephen Harrison, Steve Glassman, John DeTreville, and Marc Brown. Steve Glassman held down the fort during the months leading up to the festival. He and Stephen Harrison developed a sophisticated 2D animation package, and John DeTreville implemented GraphVBT on top of their animation package. John Hershberger ported Zeus from Modula-2 and introduced the enhancements discussed in this paper. Mike Sclafani implemented the tools for code views and data views.

The participants in the Animation Festival were Garret Swart, Jim Saxe, Dave Redell, Lyle Ramshaw, G. Ramkumar, Steven Phillips, Sue Owicki, Greg Nelson, Rustan Leino, Solange Karsenty, Anna Karlin, Bill Kalsow, Mick Jordan, Kevin Jones, Allan Heydon, Loretta Guarino, Steve Glassman, Hania Gajewska, Andrei Broder, and Yossi Azar. Their assistance (and patience) in debugging and improving the system is greatly appreciated. They were true "software pioneers."

The Animation Festival was conceived with the help of Thomas Fogarty and brought to realization with the help of Bob Taylor.

References

[1] Marc H. Brown, Zeus: A System for Algorithm Animation and Multi-View Editing, In *Proc. 1991 IEEE Workshop on Visual Languages*, pages 4–9, October 1991.

[2] Marc H. Brown and Robert Sedgewick, Techniques for Algorithm Animation, *IEEE Software*, 2(1):28–39, January 1985.

[3] Marc H. Brown, Exploring Algorithms Using BALSA-II, *IEEE Computer*, 21(5):14–36, May 1988.

[4] John D. DeTreville, The GraphVBT Interface for Programming Algorithm Animations, *In preparation.*

[5] Steven C. Glassman, A Turbo Environment for Animating Algorithms, *In preparation.*

[6] Peter A. Gloor, AACE – Algorithm Animation for Computer Science Education, In *Proc. 1992 IEEE Workshop on Visual Languages*, pages 25–31, October 1992.

[7] Samuel P. Harbison, *Modula-3*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[8] Robert Sedgewick, *Algorithms, 2nd Edition*, Addison-Wesley, Reading, MA, 1988.

[9] John T. Stasko, TANGO: A Framework and System for Algorithm Animation, *IEEE Computer*, 23(9):27–39, September 1990.