String Matching Animator SALSA

Erkki Sutinen and Jorma Tarhio
Department of Computer Science
P.O. Box 26
SF-00014 University of Helsinki, Finland

Abstract

Animation is a way to illustrate the behavior of complex algorithms and systems. We introduce the SALSA package, an animating and experimenting tool for string algorithms. SALSA was implemented using the XTango animation environment. SALSA contains animations for seven string algorithms including Boyer-Moore, Rabin-Karp, and Aho-Corasick algorithms. We also discuss the usefulness of animation for Computer Science education and research.

1 Introduction

Animation is a useful approach for Computer Science education and research. For example, the idea of an algorithm is easier to grasp by following an animation: the user may observe how the algorithm behaves with various inputs. Because animation can lead to more efficient algorithms, it is therefore to be considered a helpful interactive tool for algorithm research.

String algorithms form an important area of algorithm research. A typical problem is string matching, where approximate or exact occurrences of a pattern is searched in a text. String algorithms are applied in various areas including speech recognition, data compression, text processing, data communications, image processing, and computational biology.

Up till now, there are practically no visualizations made for string algorithms. We collected experiences in a project constructing an animation package, called SALSA, for string algorithms (the name is an acronym for String ALgorithmS Animator). The project was organized as a software engineering assignment for a team of four students of the fifth year [1].

The SALSA package is running in the Xwindows environment on Sun workstations. The implementation of SALSA was based on an animation environment XTango [4, 10, 11]: the algorithms to be animated were supplied with calls to XTango routines for creating and moving visual objects on the screen.

The rest of the paper is organized as follows. In Section 2, we will discuss the principles of the XTango environment and give an overview of different approaches to algorithm visualization. Section 3 lists the goals of the project and outlines the architecture of SALSA. In Section 4, we will discuss our experiments, concentrating on the usefulness of animation in learning and research. The final section will introduce our future plans.

In: *Proceedings of the Third Symposium on Programming Languages and Software Tools* (ed. M. Tombak), Department of Computer Science, University of Tartu, 1993, 120–129.

2 Different approaches to visualization in Computer Science

In Computer Science, visualization is used in many ways, like in drawing flowcharts, designing systems, developing user interfaces, simulating phenomena, visual programming, and teaching data structures.

In this section, we will first define the concept of algorithm visualization, according to Myers' taxonomies [5]. After that we will present the algorithm animation framework and the path-transition paradigm of the XTango environment [9, 10, 11]. Starting from this conceptual background, we will itemize the phases to produce an animation. And finally, we will consider future opportunities for perceptional exploration of algorithms.

2.1 Myers' classification of program visualization systems

Myers defines program visualization as illustrating "some aspects of the program after it is written" [5]. Visualization is code, data, or algorithm visualization according to the visualized aspect, and visualization is static or dynamic depending on the produced display.

To exemplify the taxonomy, the traditional flowchart falls into the category of static code visualization, while a graphical debugger showing the line under execution would represent dynamic code visualization. Moreover, a static data visualization system would illustrate a program's tree structure, while its dynamic counterpart would display also the changing values of the nodes.

Another example of code animation is a teaching tool ASSEM with which a user can simulate the CPU of a simple computer [7]. The user specifies the memory location of the first instruction, and ASSEM will step instruction by instruction and show the contents of the main memory and the registers.

For instance by setting the necessary parameters, it is possible to automate both the code and data visualization, without touching the code. Algorithm visualization systems, on the contrary, necessitate the programmer to explicitly add information to the code of the animated algorithm, to create an animation.

In fact, there are animation systems which can visualize the program without any additional information in the code. For instance, PASTIS animates Fortran, C, and C++ programs by making use of the debugger [6]. According to Myers' taxonomy, this kind of systems would not belong to the category of algorithm visualization. However, this is the only natural choice for PASTIS; it should not make any difference whether the animations are produced by additional code in the algorithm or, like in PASTIS, the animation modules are separated from the source code from which they get data through the debugger. The essence of (dynamic) algorithm visualization might, therefore, be defined by producing a (event-driven) visual abstraction of an algorithm.

2.2 The XTango algorithm animation environment

XTango (for XWindows Transition-based ANimation GeneratOr) is a public domain software package, delivered and maintained at the Georgia Institute of Technology.

The XTango algorithm animation environment is based on two principles: (1) the framework to map the interesting events of an algorithm to their visual counterparts, supported by (2) the path-transition paradigm, guiding the design of the animations [9, 10, 11].

The algorithm animation framework The framework consists of three components: (1) the algorithm, (2) the animation, and (3) the mapping component. In the algorithm component, the designer defines the algorithm's interesting events, or in the XTango terminology, algorithm operations. These operations correspond to the important elements of the algorithm's semantics. For instance in the string matching algorithms, the algorithm operations include at least character comparison and moving along a string. The algorithm operations are added to the algorithm as function calls.

The animation part includes the graphical objects for visualizing the algorithm, and the routines for changing their size, color, place etc. For example, to illustrate character comparison, one might specify the characters as rectangles, flash the characters under comparison, and move one on the other to show the difference. The routines for changes in the screen are called animation scenes. Although implemented at higher level by the designer, they call XTango routines to take care of the low level graphics.

The mapping component has two parts. First, XTango uses a kind of symbol table to connect a visual object with a set of parameters from the algorithm. This mechanism is called association. In our string matching algorithm example, the places of the rectangles visualizing the characters T[1], T[2],... might be stored as Assoc(ID,T,1), Assoc(ID,T,2), ...

The second part of the mapping component includes the relation between the algorithm operations and the animation scenes. In our example, the algorithm operation Character Comparison maps to a group of animation scenes: flashing and moving.

The path-transition paradigm The idea behind the path-transition paradigm is to separate the design work of the animations from the implementation phase. The paradigm supplies the designer with four abstract data types with the operations. If the designer specifies the animation scenes with these operations, the implementation should be straightforward, using the XTango routines corresponding to the abstract data type operations.

The four data types of the paradigm include (1) location, (2) image, (3) path, and (4) transition. They relate to each other as follows: An image has a location in the infinite coordinate system of the XTango window. A path is an ordered sequence of coordinate pairs, which defines relative changes in X- and Y-axis. A transition changes an image according to a path.

Let us illustrate the paradigm with an example. To design the animation scene for moving a rectangle on another, the images concerned are these rectangles. The path along which Xtango will perform the transition is defined by the locations of the rectangles.

The path-transition paradigm, in addition to the algorithm animation framework of XTango, gives the guidelines for the different phases in the design and implementation of an animation. The designer starts with identifying the algorithm operations of interest. Then, he/she will decide the animation scenes necessary

to visualize the operations; the scenes must be designed using the path-transition paradigm. The crucial phase is to define the relation between the algorithm operations and the animation scenes. Last, the implementor turns animation scenes into C functions, calling the appropriate XTango routines.

2.3 Future trends in algorithm experimentation

Among the possibilities to study an algorithm using technology, visualizing is only a beginning. However, even visualization can be of greater advantage. In the design of animation, much more attention must be paid on psychological factors. For instance, the designer should use colors in a way which helps following the animation. Brown and Hershberger list the following use of colors [2]: encoding the state of data structures, highlighting activity, tying views together (in the case of multiple windows for different aspects of an algorithm), emphasizing patterns, and making the history of an algorithm visible.

Besides the visualization, one could also make use of auralization (interpreting interesting events as sound effects) [2, 8]. As colors, they can present fundamental information on an algorithm when used for reinforcing visuals, conveying patterns (e.g. by using multiple instruments), replacing visuals (to reduce the visual information), and signaling exceptional conditions.

This is just the beginning. Maybe computer-assisted algorithm exploration only starts with seeing and hearing, leading us to smell, taste, and touch the algorithms! Virtual reality is coming inside the researcher's chamber.

3 An overview of SALSA

In this section, we will explain our objectives in the SALSA project and go through the main components of the SALSA package [1]. We will first outline the architecture of the package with comments on the choice of the algorithms.

As stated in the introduction, the SALSA package is running in the Xwindows environment. When started, SALSA opens its main window, which controls other components. The respective program module calls for the animated algorithms. These algorithms, implemented in C, include calls to the animation routines, programmed by our team; however, to control the graphics, the animation routines use the XTango functions. One of these functions opens the XTango animation window; as a result, the animations run in this window, and the user can control the animation by pushing the icons provided by XTango.

In the beginning of our project, we had no prior expertise in principles and techniques of algorithm visualization. However, we aimed at a working animation tool suitable for introductory purposes in teaching string algorithms. For these reasons, we decided to start with the very basics. Therefore, our choice comprised seven algorithms for one- and two-dimensional pattern matching, and calculating the edit distance [3].

3.1 The aims of SALSA

The overall goal of the SALSA project was to develop a computer-assisted instruction package. We intended this package to serve as an introduction to string

algorithms either in a classroom or as a self-education material. To make our aim clear, we divided it into smaller subgoals:

First of all, the package should help one to understand string algorithms of different types. For this purpose, we decided to use animation. When animating an algorithm, the student should be able to follow its behavior with various inputs and so he/she can gain insight into the essence of the algorithm.

To make SALSA as pedagogical as possible, we followed the principles of computer-assisted instruction (CAI) in the implementation [13]. This required special attention to the user interface design.

Besides the educational perspective, SALSA should also be useful for research. This means that the package should support implementation of other string algorithms. It has usually taken too much time for the researcher to design and implement an animation of an algorithm; if this phase were considerably reduced, however, the animation itself would give new ideas in analyzing the algorithm and developing it further.

In addition to animations, SALSA should also include a kind of workbench for testing the algorithms' efficiency. This is important in learning as well as in research. While animation helps in understanding the algorithm's idea, only the hard facts about the CPU time usage tell the conditions in which to apply the algorithm.

In our project, we also wanted to evaluate the usefulness of the XTango environment, although this was not particularly the aim of SALSA. We were interested especially in how efficient the path-transition paradigm would be in the animating process.

3.2 The components of SALSA

The SALSA package consists of four main components: the graphical user interface, the animations for selected basic string matching and edit distance algorithms, the CPU time measurement, and the test data generator [1].

The graphical user interface To let the user of SALSA to concentrate on the algorithms, not the package itself, it was important to make SALSA as easy to use as possible. Therefore, we decided to make the user interface graphical, implementing it with Devguide, a development tool operating in the OpenWindows environment. In the design, we followed the OpenLook standard.

Beyond the technical implementation of the user interface, it was essential to identify the suitable learning strategies supported by SALSA. It seemed quite natural to make use of processive learning. SALSA could easily take the student all the way through the different phases of the learning process: motivating, orientating, deepening, exemplifying, practicing, evaluating.

However, we can regard an algorithm also as a system which the user can simulate by perceiving the animation with varying inputs. In addition to processive learning, SALSA would support, thus, learning by simulation.

These principles in mind, we designed the main window (Fig. 1), where the user first selects the algorithm and its input, possibly setting some parameters, and then pushes the button indicating the desired function. With the function completed, the control returns to the main window, and the user may define a new procedure.

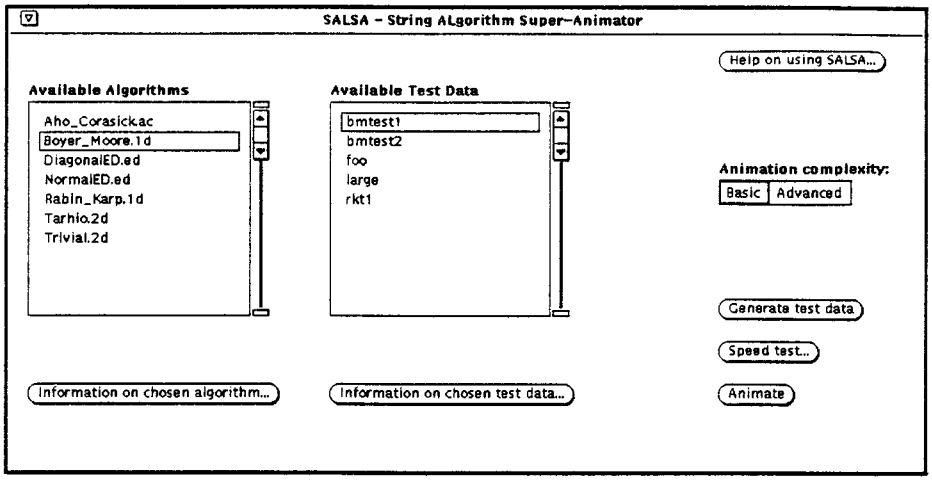

Figure 1: The SALSA main window. The user first chooses an algorithm and test data. Pushing the Animate button starts the respective animation, while the Speed test button outputs an efficiency report.

Note that it is also possible to study the algorithms only at a theoretical level, by pushing the Information button.

The animations The user can start the desired animation from the main window, pushing the Animate button. For instance, after choosing Boyer-Moore from the algorithm list and bmtest1 from the test data list, the XTango window will open and show the initial scene of the respective animation (Fig. 2).

Let us exemplify the animation procedure by looking closer at the animation of the Boyer-Moore algorithm (Fig. 3). We decided to visualize the characters by rectangles of equal width, with the height indicating the character's position in the alphabet. Second, the pattern would travel above the text. Moreover, we illustrated the comparison of two characters by moving them on one another. We displayed a match by coloring the respective rectangles black, while a zigzag arrow indicated a mismatch. A matched suffix was visualized by a line below the equal substring in the pattern.

With this design specified, we implemented the animation routines. These routines defined the locations, images, paths, and transitions, using the XTango functions.

At present, SALSA consists of the animations of basic algorithms for one-dimensional pattern matching (Boyer-Moore, Rabin-Karp, and Aho-Corasick). In addition, we implemented also the animations of calculating the edit distance with normal and diagonal methods [3]. To get insight into how a researcher may benefit from animation, we also prepared a visualization of a two-dimensional algorithm under development.

The CPU time measurement The user who is interested in the practical efficiency of an algorithm may forget the animations and run the speed test. SALSA will store the results in a log file specified by the user.

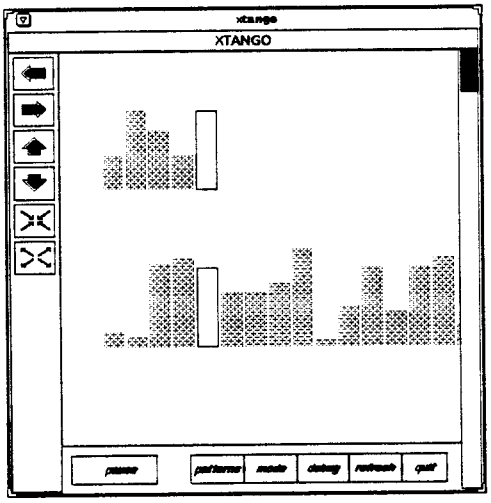

Figure 2: The initial scene of the Boyer-Moore animation. The implementor defines the visual objects inside the window by using XTango routines, while XTango provides the window with the basic operations: With the left-hand buttons, the user can pan and zoom the animation window. Moving the right-hand scrollbar downwards slows down the animation. By pushing the pause button, the user can pause the animation.

Generating and storing the test data To help the user to experiment the algorithms with various inputs, we included a test data generator SWING (for String WeavING). The user may create different inputs consisting of the desired text and patterns by connecting together two files: one containing the text, the other the patterns.

Because SALSA creates a file for each new test data generated, it is easy for the user to repeat the same run of an algorithm or to test several algorithms with the same data.

4 Discussion

Our experiences with the SALSA project produced four conclusions: First, in teaching string algorithms, animation serves as an activating teaching method which inspires students to experiment. Though rather short in code, the essence of a string algorithm is often quite hard to uncover.

A group of students of our department tested the SALSA package. The results were promising: by using the animation, it was easier to learn the idea behind the algorithm. When we made a video on SALSA, even the cameramen were interested how the algorithms worked, with no prior knowledge in Computer Science!

Second, designing an animation is a learning process which leads to a profound understanding of the algorithm. When transforming the detailed algorithm to the higher level of abstraction, the designer little by little gets closer to the essence of the algorithm. Actually, the designing process is interaction between learning, teaching, and research.

The students of our team had no prior knowledge of string algorithms. However, all of them got interested in the problem area and studied themselves the area

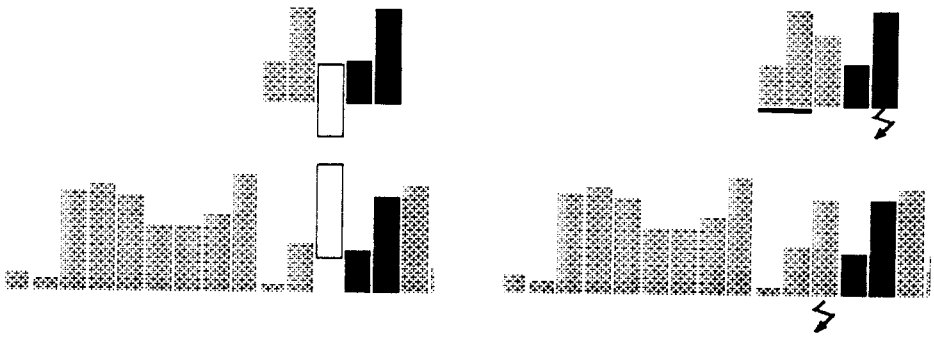

Figure 3: Two phases in the Boyer-Moore animation. In the left-hand figure, the animation illustrates the comparison of two characters, with the respective rectangles approaching each other. In the right-hand phase, the algorithm has found a matched suffix in the pattern, visualized by an underline; the zigzag arrow indicates the character responsible for the unmatch.

beyond the algorithms to be animated. What happened with the Aho-Corasick algorithm, describes well the learning process. The designer of this animation, having seen how the Boyer-Moore algorithm works, noticed that it would be efficient to combine these two methods. It was a pity that this approach is already known as the Commentz-Walter algorithm.

Third, animation is useful also for research. Experimenting with animations can indicate weaknesses in specific cases in the behavior of an algorithm under study and thus help in developing the algorithm further. Animation may also help in analyzing the complexity of the algorithm.

We got a nice example of this when animating a new algorithm for two-dimensional pattern matching. The co-operation between the animator and the researcher led to a more efficient algorithm.

In fact, it is possible to infer the aims of SALSA partly from the functions of the university: how to create a natural interaction between learning, teaching, and research. At the same time, there has been discussion about the communicative role of the university. The projects like SALSA teach the participating students to pay attention on how to present "the professional issues" for laymen.

Fourth, the path-transition paradigm of XTango proved to be an approach practicable enough for use in animating at least string algorithms. An overall evaluation of the tool was carried out during the project.

The team regarded especially the conceptual design (the path-transition paradigm) behind the XTango environment as easy to learn. For a beginner, it took about four days to implement an animation of the Rabin-Karp algorithm. The animation scenes took about 800 lines of code.

The main problem with XTango was its poor performance in the OpenWindows environment. According to John T. Stasko, the designer of XTango, the bottleneck lies in the performance of the X graphics implementation of the workstation [12].

As a consequence, the number of the images in the animation must not exceed the order of tens.

Despite other minor shortcomings, like the lack of multiple windows (to compare algorithms with one another, or to display different aspects of an algorithm), we are looking forward to the future. A new, C++ based animation environment, called Polka, is already available on the ftp.

5 Future work

Our experiences in construction of SALSA and in using animations encourage us to develop new animations. Next year we will produce an enhanced version of SALSA by incorporating animations of a new set of string algorithms. The possibility of using parallel animations and sound output will be considered.

We have also plans to make animations for other areas. There are many subject areas with hardly any animation packages, since most implementations visualize sorting, graph algorithms, computational geometry, or simulation of computer systems. One neglected area is compiling of programming languages. Many compiling techniques including parsing, attribute evaluation, and code generation are based on a parse tree. Such schemes are conceptionally easy to animate using a graphical representation of a tree.

Our promising experiences of using XTango for construction of SALSA show that production of animation packages in not any more tedious prototyping it used to be. We believe that the use of animation will rapidly increase in Computer Science education and research in the near future.

Acknowledgements

We wish to thank Prof. Esko Ukkonen for his continuous support and Prof. Kari-Jouko Räihä for introducing us the animation technology. We are grateful to Juhana Britschgi, Timo Joutsenvirta, Kai Järvenranta, and Antti-Pekka Tuovinen for implementation and fruitful co-operation.

References

[1] Britschgi, J., Joutsenvirta, T., Järvenranta, K., Tuovinen, A.-P., The SALSA animator: An animating and experimenting tool for string algorithms (in Finnish). Report C-1993-14, Department of Computer Science, University of Helsinki, 1993.

[2] Brown, M., Hershberger, J., Color and sound in algorithm animation. *Computer* 25, 12 (1992), 52–63.

[3] Gonnet, G., Baeza-Yates, R., Handbook of Algorithms and Data Structures in Pascal and C. 2nd edition. Addison-Wesley, Wokingham 1991.

[4] Hayes, D., The XTANGO environment and differences from TANGO. Electronic document included in the XTango package, November 3, 1990.

[5] Myers, B., Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing* 1990, 97–123.

[6] Müller, H., et al., The program animation system PASTIS. *The Journal of Visualization and Computer Animation* 2, 1 (1991), 26–33.

[7] Nasri, A., Computer graphics in simulating the functioning of a simple computer. *Computer Science Education* 2, 2 (1991), 161–170.

[8] Negroponte, N., Beyond the desktop metaphor. In: Meyer, A., et al., editors, *Research Directions in Computer Science: An MIT Perspective*, The MIT Press. Cambridge, Massachusetts, 1991, 183–190.

[9] Stasko, J., The path-transition paradigm: A practical methodology for adding animation to program interfaces. Electronic document included in the XTango package, College of Computing, Georgia Insitute of Technology, June 10, 1991.

[10] Stasko, J., TANGO: A Framework and System for Algorithm Animation. Ph.D. Dissertation, Technical Report No. CS-89-30, Department of Computer Science, Brown University, 1989.

[11] Stasko, J., Tango: A framework and system for algorithm animation. *Computer* 23, 9 (1990), 27–39.

[12] Stasko, J., Personal communication, 1993.

[13] Steinberg, E., Computer-Assisted Instruction: A Synthesis of Theory, Practice, and Technology. Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1991.