

A Technique for Algorithm Animation over the Web

Chittaranjan Mandal¹, Christopher M P Reade²

¹Dept of Computer Sc & Engg, IIT Kharagpur, INDIA, chitta@iitkgp.ac.in

²Kingston Business School, Kingston University, UK, Chris.Reade@kingston.ac.uk

Abstract— We discuss a novel technique for animating algorithms over the web. Although there are several existing software environments for the animation of algorithms, some of which are web-enabled, ours is designed specifically to simplify the process of adapting an algorithm for animation and delivering the animation over the web with a simple web interface. This provides a first stage for more advanced development of web-based interactions to support animation. Our goal is to provide general web-based support to enable much more widespread use of animation in teaching. In particular we want to address the active participation of the observer in using algorithm animation technology. We describe the current implementation of the animation engine which is based on a simple co-processing method with CGI implementation on a web-server, along with plans to use this as a base to include emerging technologies (web-services with XML to markup examples and asynchronous interaction). We also illustrate the current web interface with some examples.

I. INTRODUCTION

An important aid in teaching early courses in algorithms and many other process-related subjects is to animate the particular process that is being described. This is a challenging task that has been addressed many times before but solutions are still not sufficiently simple for widescale use. Regular class room teaching without computer aids can address the problem only partially, by illustrating the steps for a specific example (usually) on slides. However, such illustrations are limited when it comes to demonstrating a process with different sets of inputs. Even leading edge animation techniques usually provide a high quality rendering of only a specific problem instance. As recently noted in [1], the real benefits of animation for effective teaching come from active participation of the learner in using the animation rather than from high quality graphics. In this paper we describe a simple animation technique that provides a general solution within a limited context. It works over the web for maximum flexibility and does not require the student to have anything more than a web browser to get started. We have applied this technique to teach an initial course on ‘C’ programming to a class of nearly 700 students.

The target users were students with initially minimum understanding of the tools and techniques of programming. Some of the objectives of this work were as follows:

- avoid reliance on tools that a user may not have or may

have difficulty using at the early stages

- make it a web-based technique so that it is usable with nothing more than a web browser
- it should be interactive and should be able to work with user supplied data sets
- it should be possible for the instructor to create tutoring examples with relatively little effort

Initial learners usually have a problem with grasping the basic paradigm of programming. A good way to get them started is to walk them through simple programs. That way they can understand how the execution of a program progresses through various statements. They get to see the working of various control flow constructs and also how variables get updated. Hard-to-grasp concepts such as parameter passing and function calls can be graphically illustrated. Each of these aspects can make the learning process much easier. These aids are already available, but not readily accessible by novice programmers because they often require some initial knowledge of programming before they can be used. Another option is to provide personal assistance, but this is an increasingly scarce resource. Using a web-based solution opens up new possibilities for supporting a wider range of students of programming from novice to more advanced levels where standard algorithms for sorting and searching would be typically covered.

The rest of the paper is organized as follows. In section II an outline of the technique is given. In section III some examples are given to illustrate the technique. The underlying protocol used for the animation is described in section IV. This is followed by a discussion and comparison with related work in section V.

II. OUTLINE OF TECHNIQUE

The technique essentially relies on running a program, which is to be demonstrated, at the server. If the program should require some data, this is collected from the user’s browser and fed to the program. The trace of the program as it executes, and the outputs that are produced, are returned to the user’s browser. The program proceeds from point to point within the program. In addition to the start and end points, numerous additional points may be chosen within the program. If necessary, a point can be placed between every adjacent pair of statements.

The basic program is augmented with additional statements to do the following as it reaches a chosen point in the program.

- return display matter to the user’s browser, as appropriate. This will include information regarding the:

- outputs generated by the program
 - values stored in variables that need to be displayed
 - run-time data structures indicating currently active function calls
- collect inputs that may be required by the program
 - proceed to the next point once the user is ready or after some period of time

To make the overall scheme more practical, it should operate in user space rather than a superuser space. That way, an instructor can setup such animations without demanding special system privileges. Only a regular web server is required.

It may be noted that the user's browser does not interact with the program directly. Instead it interacts with the web server on the server – this the normal *modus operandi* for browser interaction. This system relies on the standard CGI programming interface. There are supporting methods for the following:

- start the program that is to be demonstrated
- collect display matter generated by the program and return that to the user's web browser
- collect inputs from the user that are required by the demonstration program and feed them to the program
- close the interaction session once the demonstration program has terminated

A. Algorithm encoding

The algorithm that is being animated needs to be coded in a special way so that animation is made possible. Presently, the animation is done using Perl. In the future, 'C' programs will be animated directly in 'C'. This avoids the complications of trying to emulate one language with another.

The text of the algorithm is placed in an array (@program in this case). The program starts with a call to `initInteraction()` to make necessary initializations. Thereafter, the program continues in *mini sessions* which involves opening the session, doing the required processing (which may involve some combination of collecting inputs, executing and generating outputs), and finally closing the session. Once the mini session is closed everything generated in the current session is returned to the client's browser where the animation is being viewed. A mini session is opened by a call to `openInteraction`. Helper functions such as `retrieve`, `htmlStart`, `displayProg`, `htmlFinish`, etc. assist the input/output operations. A call to `closeInteraction` in conjunction with `htmlFinish` is required to close the interaction mini session. The encoding procedure is highly mechanical in nature and may be automated.

As a next step we plan to use XML to encode example programs and animation markup. This will allow us to develop an interaction to support the example developer in the generation of the animation with as much automation as possible. This could then become a web service with an option to add the result to the animation server. To enrich the interaction, we plan to use (AJAX-style)

technology to allow the end-user more control over the layout of the animation. This is particularly important when more complex examples need to be displayed, so the user has control of the focus.

III. ANIMATION EXAMPLES

A. Comparing two numbers

First, a very simple example is presented to illustrate the working of the system in figure 1. This example compares two numbers n_1 and n_2 . Only the open screen, and one of the steps has been shown. The following features of the system are illustrated via this example.

- interactive abilities; user inputs can be accepted, outputs are displayed
- ability to display values of selected variables
- program tracing highlighting current statement in red
- buttons for stepping or running through to completion

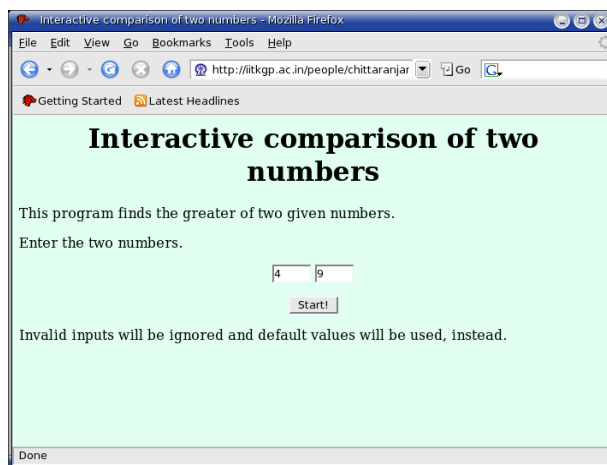


Fig. 1. Screen shot - 1 of an example to compare two numbers

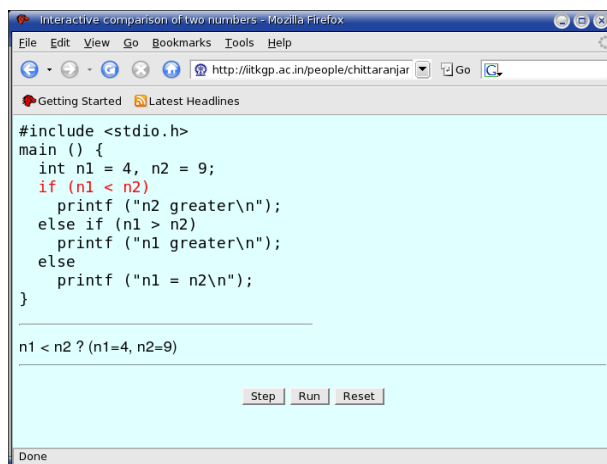


Fig. 2. Screen shot - 2 of an example to compare two numbers

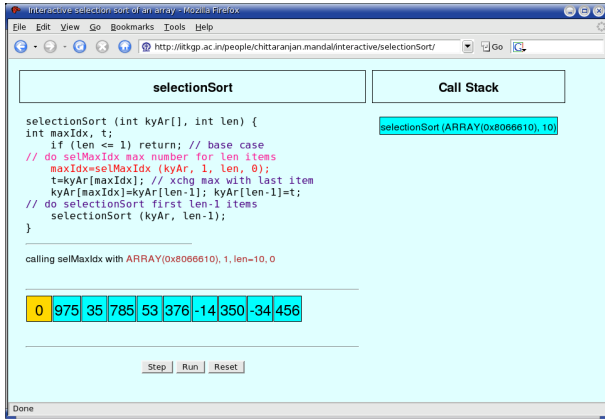


Fig. 3. Screen shot - 1 from selection sort

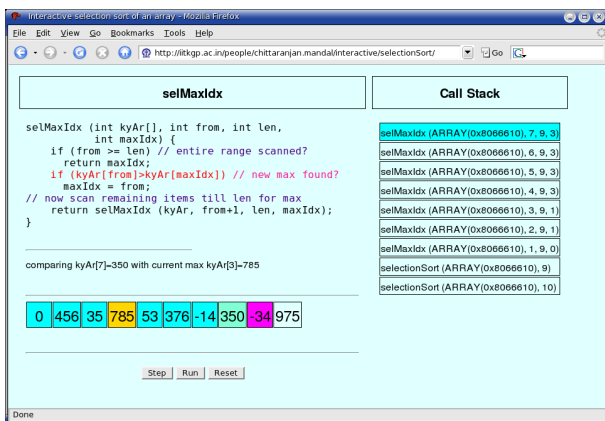


Fig. 4. Screen shot - 2 from selection sort

B. Selection sort

Next, screen shots are shown for the more complex example of selection sort in figure 3. This example illustrates additional features, such as

- function calls
- highlighting a set of statements
- distinctive highlighting of comments and program statements
- call parameters of activation records with function calls
- more complex graphics; at present all graphics are implemented by means of *HTML* constructs; this is not essential

IV. PROTOCOL FOR ANIMATION

The protocol is illustrated in figure 5. The animation is initiated and continued over the web using the CGI gateway. Each CGI request is received by the HTTP server in a stateless manner. The protocol works by storing enough state information to be able to execute the program being demonstrated; receiving inputs from the user; and returning appropriate output. The user supplies inputs via *HTML FORMS*.

There are two form actions, *startup* for initiating the animation and *carryOn* to continue with the animation

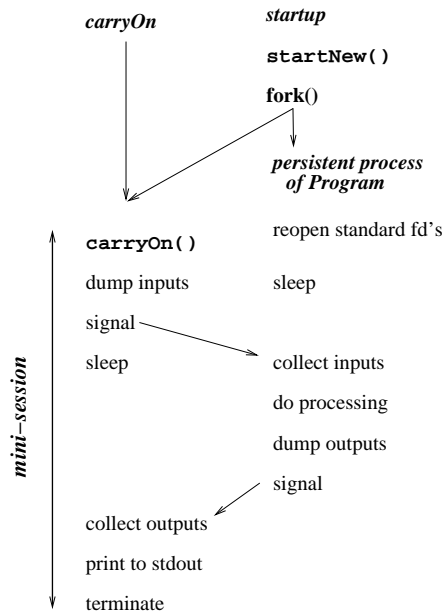


Fig. 5. Animation protocol

until it ends. The *startup* receives a path to the executable code along with the necessary inputs. A call is made to a function called *startNew* which essentially forks to create a process for the program to be animated, and itself continues to form the first mini session. After a process is spawned for animation, its first job is to reopen its standard file descriptors of *stdin*, *stdout* and *stderr*. Thereafter it goes to sleep. If it is not awakened within a certain (reasonably long) time interval it assumes that the user has abandoned the animation and it terminates.

The process participating in a mini session, continues with *carryOn*, whose job is to collect inputs sent by the user via the form; dump them into a file; signal the sleeping animation process; and then go to sleep. The process for the animation, on waking up, collects the inputs; does necessary processing; dumps necessary outputs into a file; signals the sleeping co-process of the mini-session; and then readies itself for the next mini-session by going to sleep. The co-process, on waking up, collects the output dumped by the animation process from the file; prints it to its *stdout* and terminates.

All subsequent mini-sessions are initiated by a separate script called *carryOn*. The form is submitted when the user clicks on the step button. The button click is performed automatically after a certain period of time if *Run* is enabled. This directly enters the mini-session, as shown in the diagram of figure 5 and as explained above. When the animation comes to an end, the last mini-session generates plain *HTML* output without the form. The animation process terminates once the animation comes to an end.

V. DISCUSSION AND RELATED WORK

Our goal is to provide general web-based support to enable much more widespread use of animation in teach-

ing and to support more interactive use of animations in the learning process. The implementation technique described here provides support for simple adaptation of an algorithm so that an animation can be delivered over the web for viewing and controlling via a web browser. The advantages of this approach are two-fold. Firstly, the development of an animation does not require expertise in using complex animation software and is thus a much lower hurdle for teachers wishing to produce examples. Secondly, the delivery and viewing of animations requires a minimal, platform-independent capability to be available to the end user (i.e. just a web-browser) so that the potential audience is maximized.

At the moment, only examples in the 'C' programming language are being considered, and we do not provide full automation for the process of creating animations from example 'C' programs. In future work we plan to maximize the automation using XML to markup programs and a web-service to upload examples and provide fine tuning control of the animation generation through a web interface. We also recognize the need to support more graphical displays of complex data and objects for more widespread use of the animation tool. However, we feel that a higher priority is to develop support for more interaction from the end user in running, controlling and changing animated algorithms. We believe that our simple approach to algorithm generation will enable more flexible and adaptable interactions than has been practical with traditional approaches.

Since the early days of algorithm animation, there has been an emphasis on the use of advanced graphics and support from large animation development environments (such as Zeus [2]). More recently the emphasis has returned to effective use in the learning process. With studies such as those discussed in [3] and [4] the importance of interaction rather than simple observation has been recognized. This new direction was also discussed in [1] where Jhavé was also described. Jhavé is a Java based animation tool which has been designed to take plugins for new animation techniques, and thus can act as a controlling shell for animations. This is relatively platform independent through the use of Java and can support an extensible set of interaction and animation styles. Another Java specific animation tool is Jeliot 3 [5] which has evolved from Jeliot 2000 and uses self-animating data types and is designed to allow easy uploading of examples by end-users. In [6] it is argued that smarter tools are needed to cater for and to adapt to the different abilities of users, referring to studies of taxonomies of software visualization tools.

Our tool presently used mostly HTML and just a few lines of JavaScript on the client side. It is, nevertheless, a light weight and yet capable animation tool. We have coded over thirty six animation examples (manually) and used these to teach a first-level programming course. One important advantage of our approach is that the client CPU is not significantly loaded.

REFERENCES

- [1] T. L. Naps, "Jhavé: Supporting algorithm visualization," *IEEE Computer Graphics and Applications*, vol. 25, no. 5, pp. 49–55, 2005.
- [2] M. H. Brown, "Zeus: A system for algorithm animation and multi-view editing," *IEEE Workshop on Visual Languages*, pp. 4–9, October 1991.
- [3] S. Hansen, H. Narayanan, and D. Schrimpscher, "Helping learners visualize and comprehend algorithms," *Interactive Multimedia Electronic J Computer-Enhanced Learning*, vol. 2, 2000.
- [4] C. Hundhausen, S. Douglas, and J. Stasko, "A meta-study of algorithm visualization effectiveness," *J. Visual Languages and Computing*, vol. 13, no. 3, pp. 259–290, 2002.
- [5] E. Sutinen, J. Tarhio, and T. Terasvirta, "Easy algorithm animation on the web," *Multimedia Tools and Applications*, vol. 19, no. 2, pp. 179–194, 2003.
- [6] R. Bednarik, A. Moreno, N. Myller, and E. Sutinen, "Smart program visualization technologies: Planning a next step," *Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies (ICALT 2005)*, pp. 717–721, 2005.