Integrating Animations Into Courses

Susan H. Rodger[1]
Duke University
Durham, North Carolina 27708-0129 USA
rodger@cs.duke.edu

ABSTRACT

This paper describes two ways we have integrated algorithm animations into several computer science courses. First, we use previously existing animations during lectures to aid in explaining algorithms, and second, our students write programs with animations. Different types of animations are written depending on the level of the student. Students who have never programmed before construct simple animations using an interpreted language, and more advanced students write sophisticated animation programs that are compiled.

1 INTRODUCTION

For most people, an animation of an algorithm is easier to understand than a textual or pictorial presentation of the algorithm. In an animation, key objects are drawn, visually showing their relationships, but furthermore, as the algorithm proceeds, changes in the objects and their relationships are shown through their movement. Animations of algorithms also make it easier to debug programs. If a program's output is an animation, the program's correctness is visually clear. For example, objects moving the wrong direction not only indicate a problem, but show what the problem is.

In this paper we describe how we have integrated algorithm animations into several computer science courses. Sections 2 and 3 describe how we have used animations in our courses, during lecture and outside of lecture. In Section 4 we describe Xtango[6], a tool for writing algorithm animations, and in Section 5 we discuss the types of animations students at different levels can write using Xtango. Section 6 gives an example of how we use algorithm animation in our introductory computer science course, and in Section 7 we give concluding remarks.

2 PRESENTATION IN LECTURES

We show animations of algorithms during lecture using a computer connected to either a projector or a display panel. An animation aids in both explaining and analyzing an algorithm.

Showing an animation while explaining an algorithm provides a complete picture, making it easier for students to fully understand the algorithm. Trying to animate an algorithm on a blackboard or overhead projector usually ends up in a partial picture, and a mess of erasures and overwrites. A computer animation of an algorithm allows one to neatly show the complete picture from start to finish. We control the speed of the animation, quickly moving over stages that are easy to understand in order to reach the more complicated stages, and showing these in slow motion. Pausing at a stage allows the class to discuss what just happened, and to allow students time to think about what will happen in the next stage.

Showing several animations of an algorithms on different inputs illustrates how the algorithm will behave in different cases, and aids in analyzing the running time of the algorithm. For example, we show an animation of the average behavior of quicksort on a set of data that usually partitions roughly in the middle, and then show the worst case behavior on a set of data in reverse sorted order, which shows that the partition element reduces the size of the partition by one each time.

3 EXPLORATION OUTSIDE OF CLASS

Our students use animations outside of class to explore algorithms at their own pace. Students recreate animations from the same data files used in class, viewing them repeatedly until they understand them. Furthermore, students run animations on arbitrary inputs,

[1]Supported in part by the National Science Foundation's Division of Undergraduate Education through grant DUE-9596002. The author's web address is `http://www.cs.duke.edu/~rodger`

allowing them to see how the algorithm processes different cases. For example, insertions in a red-black tree (a balanced binary search tree) are complex and difficult for many students to understand. Using an animation, students predict how the insertion will affect the tree, enter the next value to insert, and then observe the animation to check their prediction. If a prediction is incorrect, they will know immediately and are more likely to seek help. This contrasts with traditional homework assignments in which students do not receive feedback until days later. This particular animation is used heavily right before exam time.

4 CREATING XTANGO ANIMATIONS

There are many ways to create animations. One can start from scratch at a low level using X windows, Motif, tcl/Tk or some other windowing tool. Alternatively, special tools have been created specifically to generate algorithm animations, including Xtango [6], Zeus[2], and more recently Java[3]. This paper focuses on Xtango, which we have used extensively in our courses [4]. There are two ways to create an animation in Xtango, writing a program that is compiled, or generating animator commands that are interpreted.

Xtango programs are written in C using X windows and the Xtango library. In the Xtango framework, a user creates images, such as rectangles and circles, and applies transitions to them, such as moving or coloring an image. In more sophisticated animations, multiple images are manipulated simultaneously. Although the Xtango library simplifies the creation of animations, it is too complex for beginner programmers. The Xtango package comes with over 50 animations of elegant algorithms and data structures for computer science that are created in this manner.

One can also use Xtango to create animations by sending high-level commands to an interpretor called the animator. A one line command is used for either creation or movement. However, this approach is more limited than the other method as only one image at a time can be created or moved. The animator is typically used for creating animations quickly, by having the algorithm (program) output animator commands.

5 LEVELS OF ANIMATION CREATION

This section describes the types of Xtango programs written by students in computer science courses at various levels. Although using the Xtango animator is appropriate for creating quick animations by students in all computer science courses, it is best to hide the details of the animations in beginning courses. In contrast, students in upper level or project courses write more sophisticated animations by writing Xtango programs.

The animator is easy to use, even for students who have never programmed. In a half day activity we ran for high school students who mostly had not programmed before [5], students entered animator commands into a file and then fed the input to the animator. By following instructions on a handout, students created a traffic light that changed colors (from red to green to yellow, and then back to red), a road and moving cars. The cars came up to the stop light and stopped at the red light, the light turned green, and the cars zipped through. At this point the students were free to either extend this animation or create another animation. Extensions students came up with included crashing cars, ambulances, intersections, hills, mountains, and story lines.

Although the animator is easy to use as shown above, we have found that beginning programmers struggling with the syntax of a language become easily confused when they have both language error messages mixed in with animator error messages. Thus, in CPS 6, our first computer science course, students use C++ classes that automatically generate animator commands, rather than generating the animator commands themselves, allowing them to use the animator without also learning how to program it. An example animation of a hot air balloon is described in detail in the next section.

In CPS 100, our second computer science course, students create animations by writing C++ programs whose output is animator commands. In one project, students created a binary tree of integers and then converted the tree into a min-heap by swapping the appropriate values. The difficulty in this assignment was the layout of the binary tree.

For project courses and independent study courses students write Xtango programs. For example, one student developed a two-view animation for dynamically maintaining maximal points in the x-y plane (a computational geometry problem) showing the data structure in the top view and points in the x-y plane in the bottom view. Using a mouse one could insert a point in the bottom view and the change in the data structure would be animated in the top view. Every node and edge that needs to be moved, moves at the same time, for a smooth animation.

6 EXAMPLE ANIMATION IN CS 1

This section illustrates how we converted one program with textual output into an animated program in CPS 6, the first computer science course at Duke. In this course, students learn to use C++ classes early on. The first class we introduce is the Balloon class in [1]. Students create a hot air balloon, and make the balloon rise, cruise, and descend. Below is a sample Balloon program that does this.

```
main()
{
    Balloon exxon;
    int rise;         // how high to fly     (meters)
    int duration;     // how long to cruise  (seconds)

    cout << "How high (in meters) to rise: ";
    cin >> rise;

    cout << "How long (in seconds) to cruise: ";
    cin >> duration;

    exxon.Ascend(rise);       // ascend to specified height
    exxon.Cruise(duration);   // cruise for specified time-steps
    exxon.Descend(0);         // come to earth
}
```

When this program is run, students see textual information describing the current status of the balloon. In ascending, messages are printed every 10 meters as burning occurs to make the balloon rise to the specified height. In cruising, messages are printed each time step with additional random messages occuring as wind shear forces the balloon to suddenly rise or drop. In descending, messages are printed every 10 meters as air is released. A sample run is shown below.

```
How high (in meters) to rise: 60
How long (in seconds) to cruise: 14
***** (Height = 0) Ascending to 60 meters *****

0 meters  Burn! ...
10 meters  Burn! ...
20 meters  Burn! ...
30 meters  Burn! ...
40 meters  Burn! ...
50 meters  Burn! ...

***** Cruising at 60 meters with margin +/- 5 for 14 time-steps *****

60 meters (time step 0)
60 meters (time step 1)  wind-shear drop 5 meters
55 meters (time step 2)
55 meters (time step 3)
55 meters (time step 4)
55 meters (time step 5)
55 meters (time step 6)
55 meters (time step 7)  wind-shear drop 3 meters too low!  Burn! ...
62 meters (time step 8)  wind-shear bump up 4 meters too high!  Woooosh!
56 meters (time step 9)  wind-shear bump up 1 meters
57 meters (time step 10)  wind-shear bump up 3 meters
60 meters (time step 11)  wind-shear bump up 3 meters
63 meters (time step 12)  wind-shear bump up 3 meters too high!  Woooosh!
56 meters (time step 13)

***** (Height = 56) Descending to 0 meters *****

56 meters  Woooosh!
46 meters  Woooosh!
36 meters  Woooosh!
26 meters  Woooosh!
16 meters  Woooosh!
6 meters  Woooosh!
```

Manipulating one balloon and understanding the results is straightforward for most students; however, if two or more balloons are created, the reading of the output becomes tedious and students get frustrated.

As a result, we created an animated version of the program above using the Xtango animator. The main function is exactly the same as above, but now the member functions in the class generate animator commands, which animate the balloon.

The animation starts by showing the sun, grass and a balloon (dark circle) resting in the grass on the bottom right side. Calling the Ascend function makes the balloon rise, calling the Cruise function makes the balloon move to the right (random wind shear may make the balloon bounce around), and Descend makes the balloon descend. In Figure 1 the dark circle in the left top region is the balloon, which has risen from the ground and started cruising to the right.

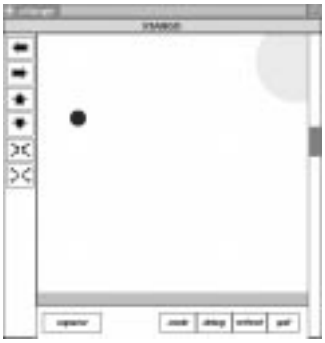

Figure 1: Balloon Animation

With the animated balloon, it is easy to see when a program is incorrect because the balloon moves in a wrong direction or not at all. More complex programs are also easier to debug when using the animator. One extension our students did was to simulate a balloon race. They created several balloons (each new balloon is automatically a different color) that rose to different heights, and then randomly moved 1-4 positions to the right over and over again. The winner of the balloon race was clearly seen.

7 CONCLUSION

Animations provide an alternative view of algorithms that is often easier to understand. Thus, we show animations in our lectures to explain algorithms by showing complete examples, and to analyze the algorithms. For the same reason, our students write animations, which provides them with visual feedback on correctness.

8 REFERENCES

[1] O. Astrachan, *A Computer Science Tapestry: Exploring Programming and Computer Science with C++,* McGraw-Hill, 1996.

[2] M. Brown, ZEUS: A System for algorithm animation and multi-view editing. *Proceedings of the IEEE 1991 Workshop on Visual Languages*, p. 4-9, Kobe, Japan, Oct. 1991.

[3] T. Ritchey, Java!, New Riders Publishing, Indianapolis, Indiana, 1995.

[4] S. Rodger, An Interactive Lecture Approach to Teaching Computer Science, *Proceedings of the Twenty-sixth SIGCSE Technical Symposium on Computer Science Education*, p.278-282, 1995.

[5] S. Rodger, and E. Walker, Activities to Attract High School Girls to Computer Science, *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education,* p.373-377, 1996.

[6] J. Stasko, Tango: A Framework and System for Algorithm Animation, *IEEE Computer*, p.27-39, Sept. 1990.