

Work in Progress: A Detail+Context Approach to Visualize Function Calls

Xiaoming Wei and Keitha Murray
Computer Science Department
Iona College, New Rochelle, New York, 10801

xwei,kmurray@iona.edu

Abstract

We describe our work in progress to use Java3D to animate dynamic function calls. Our main goal is to develop a visualization tool to help both lower level and upper level computer science students understand function call trace. A detailed visualization of a function call, such as the assignment of variables, allocation of memory, execution of control statements, etc. can help lower level students understand underlying data structures and algorithms. At the same time, a global view of the hierarchical call chain can provide greater insight for higher level students. We choose to use Java3D in consideration of visualizing larger and more complex Java programs.

1 Introduction

Software Visualization (SV) is defined as the method using typography, graphic image, sound and cinematography to demonstrate and help users understand computer software. Early in 1981, Baecker [2] presented an algorithm animation video "Sorting out Sorting". Since then, many software visualization systems or tools [1, 3, 11, 8] have been developed and evaluated by researchers based on effectiveness of teaching [4, 12].

The problem we would like to solve is the dynamic visualization of function calls. Viewing an application's call trace in graphical form can be an elucidating educational experience. For novice CS students, doing so can help them understand an application's internal behavior. Advanced students will be able to obtain information for program optimization. For example, by optimizing those functions that are called most often, you can get the greatest performance benefit from the least amount of effort. The paper is organized in the following manner. In section 2, we discuss the background and motivation for this work. In section 3, we introduce our work in progress.

2 Background

We can roughly classify SV into three categories as algorithm visualization, program visualization and the combination of the two. Algorithm visualization is the visualization of high level abstraction of data, operations and the semantics of a program. Program visualization displays the underlining source code and data structures of a program. Most of the algorithm visualizations are created by visualizing a specific data structure or algorithm. They are usually based on a higher abstraction level than the source code. Such tools are very helpful when teaching algorithm concepts, such as Quicksort and AVL-trees. For students in introductory and intermediate programming courses, we find that it is relatively easy for them to understand the fundamental algorithms in a higher level pseudo code or script language, but much more difficult to transform the pseudo code into source program. One of the reasons is that the concepts of variables, memory, function calls, etc. are hard to imagine, in other words, they can not visualize them.

We believe when teaching introductory and intermediate computer science courses, showing how an algorithm or data structure works is not good enough since the implementation of the algorithm is also important. The question is how to help students understand the dynamic aspects of programs better? Jeliot 3 [9] is an animation system that visualizes the

dynamic execution of Java programs by showing the current state of the program (e.g. methods, variables and objects) and animations of the expression evaluations. Kannusmaki et al. [6] give a thorough evaluation of using this tool in a secondary programming course. Jeliot 3 can be a great tool to teach novice students in their first programming course. However, for intermediate and advance students, an adaptive visualization tool is needed since the interests of these two groups of students are different.

As Java programs get larger, they become difficult to understand and to debug. In order for advanced computer science students to understand and further improve their programs, they have to read through the code and follow function calls for various inputs. And throughout the process, it is very easy for them to lose the global picture of the function call stack. Most of the current SV systems display the call trace either by using a fixed 2D window or using a separate pop up window for each function call. However, this leads to the problem of overlapping data for a program with many function calls. Joshi, et. al. [5] designed a tool to visualize dynamic call graph using a helix cone tree structure. The run-time behavior of executing Java programs is captured and recorded in an XML file. A viewer loads the file and renders it in the helix cone tree format. The user can also interact with the visualization result in the viewer. A static view of the function call trace will help students understand the structure of the program. However, using this approach, we will not be able to observe the internal behavior of each function. We would like to design a dynamic and step by step animation of the function call chain. In this way, novice students can interact with the current function call, while advanced students will be able to concentrate more on the global view of the call chain.

3 Current Work

We plan to use Java3D to set up a visualization environment. The approach is as follows. A global data structure is defined to represent the hierarchical structure of function calls. Each function call inserts a node into the data structure and Java3D will update the visualization interface according to the new data structure. To visually represent the hierarchical structure, we plan to use a hyperbolic graph layout representation [7]. Hyperbolic trees are defined in 3D non-Euclidean space; they are very useful and an efficient approach to visualizing hierarchical structures. Figure 1 shows such an example. While traditional methods such as paging (divide data into several pages and display one page at a time), zooming, or panning show only part of the information at a certain granularity, hyperbolic trees show detail and context at once. Students can interact with the result by rotating, selecting and scaling in the visualization interface. In the meantime, we will keep a timer to allow users to control the speed of the program. In this way, the visualization is synchronized with the running program. We will be able to animate the detailed information inside each function.

The reason to build this detail+context function visualization tool is to help both novice and experienced computer science students. Novice students concentrate more on the execution of statements and how the values of variables change. To help them, we allow the user to interactively choose one function node and display the function details in the manner of a fisheye view. Advanced students concentrate more on the overall structure of the program and care more about the complexity and how to improve and debug the program. Having a global view helps them identify loops in the function call chain easily. The students will be able to debug and optimize an application through a understanding of the call chains and their respective frequencies.

During the Spring semester, we are currently teaching a graduate level Information Visualization class. The first half of the class is devoted to a discussion of basic concepts in computer graphics, such as geometric transformation, texture mapping, lighting and shading and the introduction of techniques used in information visualization, such as parallel coordinates, cone-tree, hyperbolic tree, multi-focal display, etc. In the second half of the class, each

- [7] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. *In Proceedings of the Conference on Human Factors in Computing Systems*, pages pp. 401–408, 1995.
- [8] G. RÖßling and B. Freisleben. Animal: A system for supporting multiple roles in algorithm animation, 2002.
- [9] A. Moreno, N. Myller, and R. Bednarik. Jeliot3, an extensible tool for program visualization, 5th Annual Finnish / Baltic Sea Conference on Computer Science Education. November 17 - November 20, 2005.
- [10] Tamara Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. *In Proceedings of the 1997 IEEE Symposium on Information Visualization*, pages pp. 2–10, October 20-21 1997.
- [11] T. L. Naps, J. R. Eagan, and L. L. Norton. Jhave: An environment to actively engage students in web-based algorithm visualizations. *Proceedings of the 31th SIGCSE Technical symposium on Computer Science Education*, pages pp. 109–113, 2000.
- [12] S. Pollack and M. Ben-Ari. Selecting a visualization system. *Third Program Visualization Workshop*, pages pp. 134–140, 2004.