

AAB: A generalized Java-based Algorithm Animator Builder.

Term Project Report,

Shantanu Sharma* and Vinay Chaudhary*,

Guided by Dr. R K Ghosh**

Abstract

The project aims at building a high-level language and provides a Graphical User Interface for coding animation of Data Structures and algorithms. The software inputs commands from the application's console and stores it in its database. The interpreter then interprets the input commands and creates *objects* needed to for animation. These objects are utilized by the application and the code for animation (a Java applet code) is made based on the input commands of the user.

The software allows for demonstration of Sorting & Searching algorithms, Graph Theory algorithms and data structures like Arrays, Stacks, Queues, Lists, Trees and Graphs.

1.Introduction

An animation of data-structures is helpful to students as an educational aid in two ways, first as an alternative view in understanding a newly presented data structure or algorithm and second as an aid in debugging a program that uses the data structure. An animation can be easier to understand and remember than a textual representation, especially when one can interact with the animation by trying different input. Furthermore, using animations to debug programs can aid in finding errors faster by seeing incorrect movement of objects and/or nodes that are not connected and should be.

We have developed a Java based, easy-to-use, and architecture independent method for creating animations of data structures run over the Web. Written in Java, the program provides an interface through which users can write animations and then display them with a Web browser that supports Java. The animations are written in a simple high-level language. One does not need to know Java in order to use the software.

By writing the software in Java, users can run the animations on almost any machine and do not have to install the software locally. By leveraging the capabilities of Java and the

*Undergraduate Student, **Professor, Dept. of Computer Science & Engineering, Indian Institute of Technology, Kanpur

Web, we have built a system that will provide educators the realistic option of quickly generating algorithm animations for demonstration use in the classroom or by students generating quick animations of their programs.

The core application of the project provides a high-level language in which, the user can code for animation of algorithms. It is a convenient way to learn algorithms. However using programming languages like Java or C++ for animation-algorithms often gets cumbersome as the complexity of algorithms increases and to animate a complicated algorithm in itself becomes a difficult problem.

The project specifically addresses this problem. It provides a high level language to program the code for applets animating algorithms. The syntax of the language used is deliberately kept lucid, so that the focus of the user was on the algorithm and not on the means of animation. The syntax of the language used is given later for reference.

The input console was so designed that it prompted for almost every syntactical error in the user's code. This reduced the errors in the final code.

The only essential software needed by the project to run was the Java Runtime Environment (JRE). The project's application is executable in all operating systems which incorporate the Java Runtime Environment. Platform independence is an important feature of the application. Other than the JRE, no other software is specifically needed to use the application.

The output of the application is the code for the applet which can be compiled and the resulting applet is accessible on the internet via a web-browser.

In Section 2 we discuss related work, and in Section 3 we give a description of the application and the data-structures used. In Section 4 we describe the animation language, and in Section 5 we give examples of animating programming assignments. Section 6 describes the program design of the application, and Section 7 gives concluding remarks.

2.Related Work

Over the last decade need of packages for algorithm animation was felt in the community of Computer Science students and teachers. The packages XTango [8] and its successor Samba [9] have been used by many Computer Science Professors for a variety of topics, including operating systems [4], algorithms and data structures [7] and genetic algorithms [5].

In addition, Samba is used by students to generate animations [9].

Although the algorithm packages to date have succeeded in providing basic functionality, few have leveraged the full capabilities of the Web and Java. There is, however, a great deal of thought about the theoretical possibilities. In [6], Naps discusses how algorithm visualization could be integrated into the Web. In [3], the authors discuss ways in which the Web could be used to build courseware modules in order to aid student understanding.

Among the conclusions of both papers is the assertion that Java offers many attractive features with which an algorithm animation system could be built. There is currently much work being done in this area as summarized in [2] with most focusing on porting current applications to Java. There are also several Java applets currently on the Web that animate a single algorithm.

3. Description of the data-structures involved

In the design of the application, Object-oriented approach was used. Objects of type *Number* (a decimal number), *NumberArray* (an array of numbers), *Circle*, *Rectangle*, *Line*, *String* and *Button* (all graphical entities with associated font, colour, and dimensions attributes) were made. This greatly simplified the user's task as these graphical entities could be used as elements of a heap or stacks (& queues) or as nodes of linked lists, trees or directed (& undirected) graphs.

The software is capable of animating the following data structures:

Vectors, Arrays, Stacks, Queues, Linked Lists, Heaps, Trees, B Trees, Directed Graphs and Undirected Graphs.

These objects are discussed below:

Primary objects include lines circles, text, rectangles, and number-objects. Secondary objects include number arrays and arrays of circles, rectangles and number-object. These objects can be constructed by using their respective dialog boxes obtained from the Geometry option available in the menu bar of the application. The attached snapshots describe the application briefly. It shows the menu bar and the dialog box for creating a number object. The possible fields for enclosure are rectangle or a circle.

For creating the data structures like arrays, stacks, queues, heap, graphs etc. the secondary objects were used. Note: the secondary objects also have an associated graphical attributes of font, colour, size etc. Linked lists could be implemented by the use of primary data structures. In the dialog box for a graphical entity there are fields for all the attributes of the entity. On starting a new animation of an algorithm, the user specifies the attributes of these graphical entities and later uses them. All the commands of the algorithm are inserted in the in the Event Console. Thus the user can manipulate the attributes of these objects while coding the algorithm, by using appropriate commands (like CC, MV) at the command prompt. The language for used for coding is discussed later on.

The following describes the implementation of the data structures:

Arrays:

Arrays are the in build data structures in the application. The user can specify the attributes of number arrays in the NumberArray dialog box. They have an associated font size, number's colour, enclosure (rectangle/circle)

etc. These arrays form the building blocks of other data structures like heaps, graphs etc. NumberArray has an option to specify the x and y spacing of the constituent NumberObjects.

Trees:

Trees can be created by specifying the position of each node (number-objects), or by defining an array of number objects. *Line* graphic object is used to define an edge in the tree. The attributes of the trees can be specified by a similar procedure as for arrays. After implementing the tree data structure, the user can use it for implementation of advanced data structures like Red-black trees-trees, AVL trees are build by assigning desired attributes to the constituent objects.

Graphs:

Graphs in the application can be drawn with node placements specified by the user in the dialog box. To build a specified layout graph, the data structure NumberArray is used. The location and size of a node is also specified in the dialog boxes. Once the nodes are made, the LineArrays are used to define the edge set of the graphs. An attribute isDirected of Line and LineArray object is used to distinguish between directed and undirected graphs.

Data structures like Heaps, Stacks and Queues are generated by the NumberArray objects.

4. Architecture of the Application:

The overall design of the application can be broadly classified into the following categories:

1. The graphical user interface (GUI): the buttons and menu-items for creating the geometrical entities, editing options and incorporating subroutines.
2. A Console for input from the user.
3. A Translator converting the user's code into a code that can be easily processed by the application.
4. An application to manage the database of the variables declared by the user and their corresponding variable name used by the application.
5. An Interpreter to convert the application-code to a Java Applet program.
6. Error Console to enumerate the errors done by the programmer while coding.
7. Algorithm Output: to monitor the progress of the algorithm while coding.

Graphical User Interface (GUI):

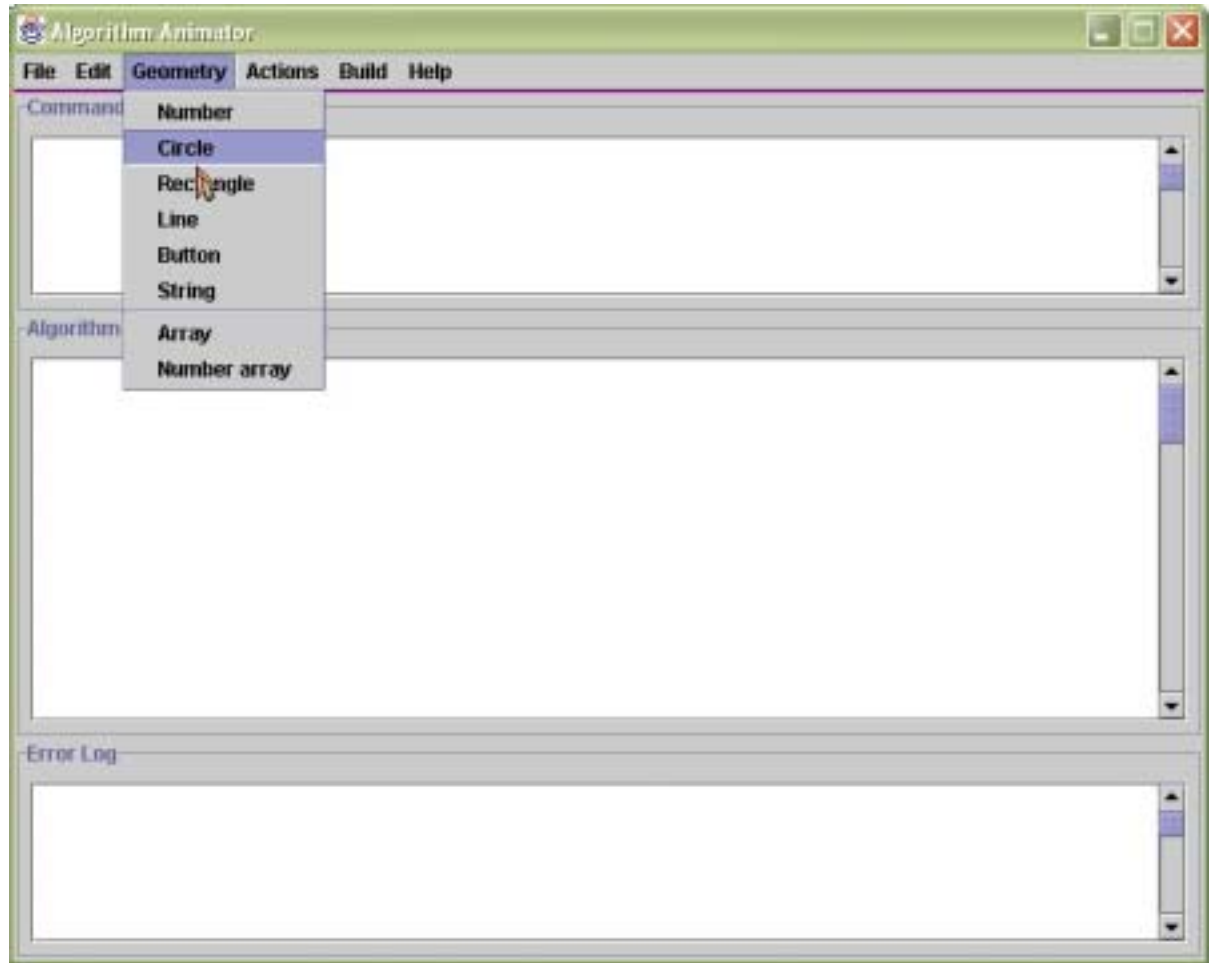
Java Swing was used for making the GUI of the application. As the application is object-oriented, it associates graphical attributes (like font, coordinates, colour etc.) of a graphical object (e.g. a text or a line) with the object itself; the menubar had options to declare the objects, specifying their attributes. (A dialog box derived from the Java class: JDialogBox was used. The objects that are possible to be created by the application are numbers, strings, array of numbers, string array and graphical objects like circle, rectangle etc. Other variables (which don't have associated attributes like font, colour etc) like integer could be directly declared in the application's console. Building of the applet's code and its compilation was possible within the application's GUI.

The menu bar had options to declare objects used in the program, dialog boxes for editing e.g. to insert a command after a given line in the algorithm or to delete a line etc. (the actual coding of the algorithm was done in the command prompt.)

In the Actions tab, it had options to declare a subroutine (used for implementing recursive algorithms) and also to declare a local variable (of the type NumberObject, Circle, Rectangle, Line String etc. and arrays of these objects).

The input for the application could be directed from a file in order to reduce the work of the user. The option for directed input was in the edit tab of the menu bar.

The following is a snapshot of the main application window:



The actual algorithm sequence was displayed in the "Algorithm Output" tab. The steps of the algorithm were numbered so as to give a better idea of the algorithm. In case of any syntactical error in the input the "Algorithm Output" was not updated and the "Error Log" shows error in the user's input with suggestions to correct it. The task of taking only the correct input from the user was managed by "EventInterpreter" class.

On clicking on the "Number" menu the following dialog box takes the input from the user:

Give number specifications:	
Variable Name: (Optional)	number
Enter the number here:	100000
Specify the font-size:(default 30)	30
Enter the coordinates here:	
20000	20
Choose enclosure	
Choose Colour	Choose Enclosure's Colour
OK	Cancel
<input type="checkbox"/> Show in the first page	

Console:

The console is a text-area build by the Java class JTextArea which could input the code (in the language provided by the application) from the user. The job of the console is to take the input from the user and gives it to the translator.

Translator:

The translator converts the console input into a code which can be conveniently processed by the software for further use and stores it in a file (pseudoCode.txt). It has an addition task of keeping a close vigil on the syntax of the console input and to prompt back to console in case of erroneous input. This helps in reducing the errors in the final code.

The translator also updates two files, one of which has the description about the objects used in the program pseudoObjectCode.txt, and another has the database of variables defined by the user and their corresponding variable names used in the program. This was needed in order to maintain consistency during the parsing of translator output.

Data base storage:

It keeps track of the variables defined by the user and their corresponding variable names used in the program. Keeping separate variable names in the program was needed by the *Input Parser*. The database storage mechanism was built in the translator. It stores all the variables in the file DataBase.txt

Input Parser:

It is the application which takes the code generated by the *Translator* and the database as input and converts it to Java applet code. As object oriented approach is used for creating numbers, circles rectangles, instances of these objects were made by the input parser whenever needed in the program. To avoid flickering in the applets, "off-screen double buffering" was used. As the code for double buffering

was generic it was added to every output file (applet code). The pseudoCode.txt was parsed and corresponding output file, i. e. the applet code was generated. Actions for commands like MV, CC were defined in the input parser.

Thus we get the final code for the applet, which could be then compiled and executed to animate the algorithm.

5. Animation Language

The animation language has been designed to require little programming experience. Animation scripts have a simple format of one command per line. There are two types of commands, those that handle specific objects and those that specify an action on an object. Of those that deal with specific objects there are two types of objects that can be manipulated, primitive objects, such as a circle, and secondary objects. Secondary objects represent data structures.

The following fields are defined by the language:

1. NUMBER: An integer or float variable which is used as a graphical object in the applet. It has a font-size, font-colour, x & y coordinates and an enclosure associated with it. The enclosure can be either a rectangle or a circle, and has a fillcolour (*Note*: colour of number is the font-colour, which is different from the colour of the enclosure). In this manner integers and float variables are given a graphical representation.
2. NUMBERARRAY: An array of NUMBER-objects. It has an associated x, y coordinate spacing of NUMBER-objects besides the other attributes of NUMBER.
3. CIRCLE: A graphical object used to denote a node in graph theoretic algorithms. Has attributes of centre-coordinates, radius and fill-colour.
4. RECTANGLE: A graphical object used for creating boxes, which are needed to clarify the demonstration of applets. Has attributes of top-left corner's coordinates, the width and height of the rectangle and fill-colour.
5. LINE: A graphical object used to represent the edges of trees and graphs. It has associated attributes of the initial and final coordinates and fill-colour.
6. STRING: A graphical object used to insert a message or a title in the applet. Associated attributes were string text, font-size, font-colour, enclosure-type, enclosure colour and the coordinates of the string.
7. BUTTON: A graphical object used to make buttons in the applet, which could listen to commands like (Mouse clicked) and perform necessary actions. Attributes were: label of the button, and the coordinates of the button in the applet.

8. ARRAY: To represent an array of the above graphical entities. Has an associated size and coordinate spacing between the objects.

The following commands are defined in the language:

1. IF (<condition>) THEN
 <Code>
 ELSE <Code>: for an "if-then-else" conditional in the program.
The ELSE clause is optional. The <condition > must evaluate to a Boolean value
2. WHILE (<condition>) DO
 <Code>: for a "while" loop in the program.
The <condition > must evaluate to a Boolean value
3. FOR (<initialization>, <condition>, <iteration>)
 <condition>: for a "for" loop in the program.
The <condition > must evaluate to a Boolean value
4. CC <object> <color> : to copy the colour specified by <color> (e.g. blue) into object's colour.
5. MH <object> <final x-coordinate>: to move the object horizontally to final x coordinate = <final x-coordinate>.
6. MV <object> <final y-coordinate>: to move the object vertically to final y coordinate = <final y-coordinate>.
7. MVH <object> <final x-coordinate, final y-coordinate> : to move the object vertically to final x-coordinate followed by moving to final y-coordinate, i.e. moving vertically and then horizontally.
8. MHV <object> <final x-coordinate, final y-coordinate> : to move the object vertically to final y-coordinate followed by moving to final x-coordinate, i.e. moving horizontally and then vertically.
9. MST <object> <final x-coordinate, final y-coordinate>: to move the object in a straight line to <final x-coordinate, final y-coordinate>.
10. SWAP <object1> <object2>: to swap object1 with object2.

In addition Subroutines are also incorporated in order to animate recursive algorithms like Depth First Search, Minimum Spanning Tree etc. To declare a

subroutine and to implement the actions of the subroutines there is a dialog-box in the menu bar.

6.Coding Procedure:

The user declares the variables to be used by the dialog boxes of the objects from the Geometry tab of the menubar. These variables are taken as global variables in the program. In order to define local variables, the Actions tab has an entry for declaring the local variables (all the fields defined in the languages can have global and/or local (uniquely) appearances. At this stage the files pseudoObjectCode.txt and pseudoEventCode.txt are updated in order to store the database and the attributes of objects.

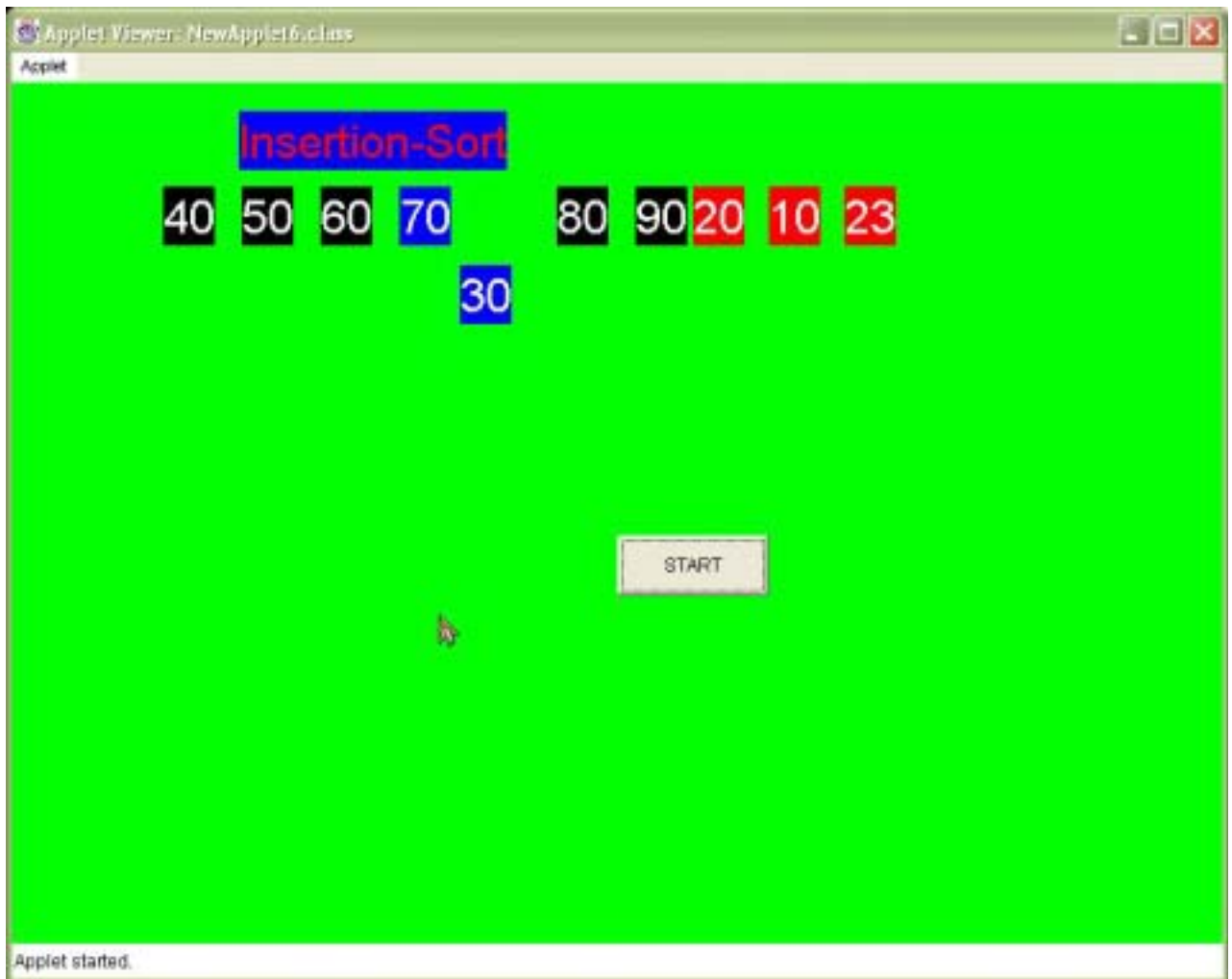
The code for the algorithm with the desired animation is then input either from a file or from the application's command prompt. In case of any error the error log shows a description of the error. A non erroneous input is shown in the Algorithm console and correspondingly file having the pseudocode of the algorithm (pseudoEventCode.txt) is updated. In order to declare subroutines the subroutine dialog box is used. In case the input is complete and syntactically correct, the InputParser parses the pseudoEventCode.txt and writes the corresponding applet code in the file NewApplet.java. A sample the code of insertion-Sort is shown below

Insertion-Sort: User's input

```
int index
FOR (index = 1 ; index <10 ; index = index +1 )
key.a = array[index]
CC key.e BLUE
MV key 150
int i
i = index -1
CC array[i].e BLUE
WHILE (i >=0 && array[i ] > key) DO
MH key key.x - 40
MH array[i] array[i].x + 40
CC array[i].e BLACK
IF (i >0) THEN
CC array[i-1].e BLUE
END
array[i+1].a = array[i]
i = i -1
END
MV key 100
array[i+1].a = key
IF (i >=0) THEN
CC array[i].e BLACK
END
CC array[i+1].e BLACK
```

```
END
WAIT START
IF START
END
  FOR (index = 1 ; index <10 ; index = index +1 )
key = array[index]
CC key.e BLUE
MV key 150
i = index -1
CC array[i].e BLUE
```

Sample Output:



Database entries:

File Name: NewApplet.java

NUMBERARRAY NA1 10 30 0 100.0 100.0 (255,255,255) 40.0 0 (255,0,0)

true 95 99 83 11 30 44 59 17 93 34

NUMBER N1 0 30 20.0 20.0 (255,255,255) 1 (255,0,0) false

INTEGER j

INTEGER i

Insertion-sort: Translator's output

int j

FOR (j = 1 ; j <10 ; j = j +1)

N1 = NA1[j]

CC N1.e BLUE

MV N1 150

int i

i = j -1

CC NA1[i].e BLUE

WHILE (i >=0 && NA1[i].n > N1.n)

MH N1 N1.x - 40

MH NA1[i] NA1[i].x + 40

CC NA1[i].e BLACK

IF (i >0) THEN

CC NA1[i-1].e BLUE

END

NA1[i+1] = NA1[i]

i = i -1

END

MV N1 100

NA1[i+1] = N1

IF (i >=0) THEN

CC NA1[i].e BLACK

END

CC NA1[i+1].e BLACK

END

7. Conclusions:

Algorithm Animator is a Graphical user interface with a high level language for creating animations of data structures and algorithms. Its output can be compiled and run on any platform with the Java Runtime Environment to generate an animation.

When used in conjunction with traditional teaching methods, Algorithm Animator can provide students with an alternative and visual perspective, which may help increase their understanding.

References

- [1] A. Badre, C. Lewis, and J. Stasko, Empirically Evaluating the Use of Animations to Teach Algorithms, *Proceedings of the 1994 IEEE Symposium on Visual Languages*, p. 48-54, 1994.
- [2] C. Boroni, F. Goosey, M. Grinder, R. Ross and P. Wissenbach, WebLab! A Universal and Interactive Teaching, Learning, and Laboratory Environment for the World Wide Web, *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, p. 199-203, 1997.
- [3] D. Carlson, M. Guzdial, C. Kehoe, V. Shah and J. Stasko, WWW Interactive Learning Environments for Computer Science Education, *Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, p. 290-294, 1996.
- [4] S. Hartley, Animating Operating Systems Algorithms with Xtango, *Twenty-fifth SIGCSE Technical Symp. on Computer Science Education*, p.344-348, 1994.
- [5] D. Jackson and A. Fovargue, The Use of Animation to Explain Genetic Algorithms, *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, p. 243-247, 1997.
- [6] T. Naps, Algorithm visualization served off the World Wide Web: why and how, *Proc. on Integrating Technology into Computer Science Education*, p.66-71, 1996.
- [7] 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.
- [8] J. Stasko, Tango: A Framework and System for Algorithm Animation, *IEEE Computer*, p.27-39, 1990.
- [9] J. Stasko, Using Student-Built Algorithm Animations as Learning Aids, *Twenty-eighth SIGCSE Technical Symp. on Computer Science Education*, p. 25-29, 1997.
- [10] D. Tunkelang. A practical approach to drawing undirected graphs. Technical Report, Carnegie Mellon University, June 1989.
- [11] HiroYuki Watanbe, Heuristic graph displayer for g-base. *International Journal of Man-Machine Studies*, p. 287-302, 1989.
- [12] J. Wilson and R. Aiken, Review of animation systems for algorithm understanding, *Proceedings on Integrating Technology into Computer Science Education*,p.75-77, 1996.