SOFTWARE VISUALIZATION: USING PERCEPTUAL, ATTENTIONAL, AND COGNITIVE CONCEPTS TO QUANTIFY QUALITY AND IMPROVE EFFECTIVENESS

by

PHILIPPA M. RHODES

(Under the Direction of Eileen Kraemer)

ABSTRACT

Software visualization (SV) involves the use of the crafts of typography, graphic design, animation, and cinematography with modern Human-Computer Interaction technology to facilitate both the human understanding and effective use of computer software. Software visualizations are often used to portray both concrete and abstract concepts and range from depictions of source code to performance characteristics to the execution of an algorithm as a discrete or continuous sequence of graphical images, or algorithm visualization. Numerous algorithm visualizations have been developed for use in educational settings. However, studies that were designed to demonstrate the pedagogic effectiveness of algorithm visualizations have been markedly unsuccessful, in spite of high expectations. In response to these results, lists of recommended features have been suggested to algorithm visualization system designers, but most of these features have not been proven to be beneficial.

The broad goal of this research is to provide an empirically-validated method for designing and evaluating the effectiveness of dynamic visualizations. Our approach has been to identify features of these visualizations and systems that may improve learning, to create software that can isolate features of interest and aid in evaluating the usefulness of these features, and to then use the software to conduct and analyze user studies.

This research:

i) assembles an initial listing of features of SVs and SV systems and introduces a framework for testing the effectiveness of each,

ii) provides verified design guidelines for dynamic visualizations,

iii) applies concepts already researched and established in perceptual psychology and cognitive psychology to the design of effective SVs,

iv) offers an explanation of the inevitable variability present in studies involving human subjects through the investigation of the effects of individual differences on comprehending SVs, and

v) objectively classifies SV systems and makes them widely and easily available in a way never done before, with VisIOn, an Interactive Visualization Ontology.

INDEX WORDS: Software Visualization, Program Visualization, Algorithm Visualization, Algorithm Animation, Cognitive, Human-Computer Interaction, Human Factors, Usability, User Studies, Experimentation, Ontology

SOFTWARE VISUALIZATION: USING PERCEPTUAL, ATTENTIONAL, AND COGNITIVE CONCEPTS TO QUANTIFY QUALITY AND IMPROVE EFFECTIVENESS

by

PHILIPPA M. RHODES

B.S., North Carolina Agricultural and Technical State University, 1997

M.Ed., The University of Georgia, 1998

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2007

SOFTWARE VISUALIZATION: USING PERCEPTUAL, ATTENTIONAL, AND COGNITIVE CONCEPTS TO QUANTIFY QUALITY AND IMPROVE EFFECTIVENESS

by

PHILIPPA M. RHODES

ACKNOWLEDGEMENTS

I would like to thank my committee members, family, friends, fellow students and most importantly God!

Thank you to my major professor, Dr. Eileen Kraemer, for sharing her invaluable advice, guidance, and wisdom and for providing both mental and physical nourishment throughout the years. Dr. Elizabeth Davis, Dr. Thiab Taha, and Dr. Paul Schliekelman, I greatly appreciate you spending time to meet with me to discuss this research and provide input from your respective areas of expertise.

I would like to thank the other graduate students from the VizEval lab and project whose hard work continuously motivated me – Matt, Sujith, Bina, Ashley, Shradha, Manish, Shaohua, and Ken. Thank you to the Computer Science Department's staff, Jean Power, Elizabeth Williams, and Claudia Sewell, for always making sure that proper documents and processes were completed correctly and on time.

None of this would have been possible without the love, support, encouragement, and shared faith of my family. Mommy and Daddy – Thank you! Paula, Stefon, and Tiffany – Thank you! The Shipman Family – thank you for your infinite prayers and strong commitment to education.

Thank you to my friends who were with me in person and in spirit during this journey. Thank you for believing in me, understanding me, and helping me to maintain some sanity.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS iv

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

1 INTRODUCTION 1

1.1 The Problem 1

1.2 Research Questions 2

1.3 Methodology 3

1.4 Contributions 6

1.5 Overview 8

2 REVIEW OF THE LITERATURE 10

2.1 Algorithm Animation Background 10

2.2 Visualization and Psychology 11

2.3 Studies of Individual Features 13

2.4 Algorithm Animation System Design 14

2.5 Summary 20

3 FRAMEWORK AND METHODOLOGY 22

3.1 Introduction 22

3.2 Just Noticeable Difference Software 23

3.3 VizEval Suite 28

3.4 System to Study the Effectiveness of Animation 34

3.5 Conclusion 36

4 PERCEPTUAL AND ATTENTIONAL STUDIES 38

4.1 Introduction 38

4.2 Noticeable Difference Study 39

4.3 Effect of Flash Cue on Detection and Localization 41

4.4 Cueing and Motion in Algorithm Animation Comprehension 46

5 ATTENTIONAL AND COGNITIVE STUDIES 52

5.1 Introduction 52

5.2 Interactive Questioning in Algorithm Animation Comprehension, Part I 53

5.3 Interactive Questioning in Algorithm Animation Comprehension, Part II 61

5.4 Eye-Tracking Study I: Correlation Between Attention, Comprehension, and Individual Differences 74

6 SOFTWARE VISUALIZATION ONTOLOGY 78

6.1 Introduction 78

6.2 Background 79

6.3 Software Visualization Ontology Schema 83

6.4 VisIOn: Interactive Visualization Ontology 88

6.5 Vision for VisIOn 98

7 CONCLUSION 100

7.1 Summative Analysis 100

7.2 Limitations....101

7.3 Conclusion and Future Work 101

REFERENCES 104

APPENDICES 112

A Sample Consent Form and Handouts....112

B Code for Version of Quicksort Used in Studies....119

C Sample Pre- and Post-Tests 121

D Sample Pop-up and Traditional Questions 127

E Quick Glance of Algorithm Animation Studies by Feature....134

LIST OF TABLES

Page

Table 4.1: 2 x 2 Factorial Experiment Design 47

Table 4.2: Averages for Pop-up Questions 50

Table 4.3: Averages for Comprehension Questions 51

Table 5.1: 2x3 Factorial Experiment Design 54

Table 5.2: Average percentage correct per group on the "traditional" test questions....57

Table 5.3: Average percentage correct per group on the "pop-up" questions 58

Table 5.4: 2x3 Factorial Experiment Design 62

Table 5.5: Pre-test scores 67

Table 5.6: Post-test (subset) scores 67

Table 5.7: Improvement as a difference between pre-test and a subset of post-test scores....67

Table 5.8: Post-test scores for all questions 67

Table 5.9: Improvement and post-test scores by Learning Style 69

Table 5.10: Interactive question scores by Learning Style 71

Table 5.11: Pearson Product-Moment Correlation, *r*, for All Tests 73

LIST OF FIGURES

Page

Figure 1.1: Visual overview of our research procedures and contributions 9

Figure 2.1: A representation of the "Learning Pyramid" 18

Figure 3.1: Screenshot of the set-up window of JND 24

Figure 3.2: A depiction of an overhead view of a participant during an experiment session 25

Figure 3.3: The color palette screen of JND 25

Figure 3.4: Prompt to guide viewer's focus to the same location at the beginning of each trial ...26

Figure 3.5: Sample display and question 26

Figure 3.6: Sample logfile produced by one session of an experiment using JND 27

Figure 3.7: The VizEval Architecture 29

Figure 3.8: Screenshots of TestCreator 31

Figure 3.9: Sample test file created by TestCreator and read by TestTaker 32

Figure 3.10: The TestTaker interface during a session 33

Figure 3.11: Sample log file created by TestTaker 33

Figure 3.12: Screenshot of SSEA 35

Figure 4.1: The TestTaker interface during a session 43

Figure 4.2: Screenshot of the SSEA program with a pop-up question displayed 48

Figure 5.1: Screenshot of the SSEA program with a pop-up question displayed 55

Figure 5.2: Distribution of Learning Styles Scores 69

Figure 5.3: SSEA – Eye-tracker Areas of Interest 76

Figure 6.1: The Software Visualization Ontology Schema 84

Figure 6.2: Software Visualization Users 89

Figure 6.3: The VisIOn Architecture 92

Figure 6.4: The VisIOn Homepage 93

Figure 6.5: The VisIOn Quick Search Page 94

Figure 6.6: The VisIOn Advanced Search Page 95

Figure 6.7: The VisIOn System Properties Page 96

Figure 6.8: The VisIOn Feedback Page 97

Figure 6.9: The VisIOn "Add a System" Page 98

CHAPTER 1

INTRODUCTION

The field of visualization has been described as a discipline concerned with the use of graphical representations in the computing environment [69]. Software visualization (SV) involves the "use of the crafts of typography, graphic design, animation, and cinematography with modern HCI *(Human Computer Interaction)* technology to facilitate both the human understanding and effective use of computer software" [60]. Software visualizations are often used to portray both concrete and abstract concepts and range from depictions of source code to performance characteristics to the "execution of an algorithm as a discrete or continuous sequence of graphical images", or algorithm visualization [27]. Algorithm animations (AAs) are dynamic algorithm visualizations. Numerous algorithm animations have been developed for use in educational settings and claims of the pedagogical benefits of these tools have been made since they were first brought into existence decades ago [5, 9].

1.1 The Problem

Assumptions that visualizations are better than textual descriptions and that dynamic displays are better than static displays have led some educators and researchers to believe that students who use algorithm animations should understand concepts faster and more accurately than those who use only textual and/or static depictions [13, 30]. These assumptions have been questioned and tested by a number of researchers [13, 30, 38, 44, 45, 87]. Unfortunately, as expressed in [36], the results of these studies and others have "yielded mixed results" and have "fail(ed) to substantiate" the belief that these tools are better than alternate forms of learning. Gurka and Citrin [28] state that these studies that were "designed to demonstrate the pedagogic effectiveness of algorithm animation programs have been markedly unsuccessful, in spite of high expectations."

Developers often build AAs based on their own opinions and expectations of use rather than on studies of student behavior [81]. Some of these developers mention the lack of a rigorous, objective method for evaluating the effectiveness of their systems, but acknowledge that creating such a method would be very difficult and complex [9, 60]. Lists of recommended features have been suggested to AA system designers [55, 71], but most of these features have not been *proven* to be beneficial. Few empirical studies have been conducted to attempt to gain insight into the effect of specific animation features on learning an algorithm [44, 73], and there has yet to be a comprehensive, empirically supported method proposed for designing algorithm animations and systems and for evaluating their effectiveness.

1.2 Research Questions

The mixed results of the previously mentioned studies have shown that some versions of software visualizations enhance learning, as indicated by students' test scores, while other studies have revealed that students using SV systems had no advantage over their counterparts who learned via the more traditional methods of lecture and/or reading text with static images. *Why?* Are the visualizations and systems appropriately designed? Were the color, shape, size, and number of graphical objects selected to maximize the viewer's "cognitive economy" [85]? Can the user control the speed, granularity, or input set? These are the types of questions investigated in the first portion of this research.

RESEARCH QUESTION: *How do perceptual, attentional, and cognitive concepts contribute to effective animation design?*

Algorithm animations have been developed since the early 1980s and currently hundreds, maybe thousands, exist. So, how does an instructor efficiently find and select an animation or system that is relevant to a particular topic? Will that tool be beneficial to students who interact with it? Can it be seamlessly integrated into the curriculum of a particular course? Was it empirically evaluated? Is there any literature describing its architecture, capabilities, and installation requirements? Answers to these questions are not readily available for most SV systems, further contributing to the lack of success of these tools and the motivation behind the other part of this research.

RESEARCH QUESTION: *How can the effectiveness of software visualizations be evaluated in an objective, systematic manner?*

1.3 Methodology

The general problems that this research addresses are the design and evaluation of effective dynamic visualizations. In particular, we focus on visualizations and visualization systems designed for computer science education. Our approach has been to identify features of these visualizations and systems that may improve learning, to create software that can isolate features of interest and aid in the evaluation of the usefulness of these features, and to then use the software to conduct and analyze user studies.

Appropriate graphical representations of concrete and abstract concepts and attributes such as color, shading, dimension, size, shape, texture, and motion are some of the types of characteristics considered when creating effective visualizations. Mackinlay [47] explains that effectiveness should be determined by whether a graphical language takes full advantage of the capabilities of the output medium and the human visual system. The effectiveness of a visualization should also be evaluated based upon its purpose. For instance, viewers of scientific displays are often trying to explore data to make inferences that could possibly lead to some type of breakthrough, so evaluations should consist of a way to measure the visualization's ability to lead the scientist to such observations [72].

Information visualization (IV) is "the use of computer-supported, interactive, visual representations of abstract data to amplify cognition" [14]. Therefore, effectiveness in the IV domain may be based on the amount of time needed to interpret data represented by images. A dynamic visualization used for pedagogical purposes, such as an algorithm animation, should be evaluated based on its ability to accurately convey the intended concept to a student in a manner that is intuitive and requires less effort than other approaches to conveying those same concepts.

A software visualization system is a web-based or stand-alone application used to specify, execute, and control software visualizations. The definition of effectiveness of a visualization system can also vary depending on the environment in which it is used. Software systems in the bioinformatics and information visualization domains need to be capable of displaying large amounts of data in a meaningful, scalable manner. In the software visualization domain, we define *effectiveness* of a *system* to be how well and easily a system can be used by the software visualization designer to communicate information to the software visualization viewer. For a designer, an effective SV system would be one that allows her to efficiently specify what and how information is to be displayed and manipulated. Speed control, zooming, and the ability to input data are the types of system features that allow a viewer to interact with a visualization and should also be considered when evaluating the effectiveness of an SV system.

Designing and evaluating algorithm animation systems are highly interrelated tasks. They both involve "de-featuring" the visualizations or systems and then establishing the level of usefulness of each feature. We have conducted experiments that isolate a feature and measure its effect on the comprehension of an algorithm. These may be perceptual/attentional features (i.e. color, motion, etc.) of the animation or attentional/cognitive features (i.e. various types of interaction) offered by the software visualization system.

The results from these studies are then used to develop design guidelines for creating more effective AAs and systems for use in educational settings. These same studies also assist in determining *to what extent* each feature contributes to the viewer's ability to perceive and comprehend the depicted algorithm and can, in turn, assist in providing a validated method for evaluating an entire visualization or system.

When creating dynamic visualization systems, one must consider both the design of the animation and the functionality provided by the system to support user engagement. This research investigates perceptual, attentional, and cognitive features through studies that focus on low-level characteristics and types of interaction as well as their influence on overall algorithm comprehension. Through collaboration with perceptual psychologists, software packages have been developed [42, 65] that we are using to create, conduct, and analyze these experiments.

Evaluating the effectiveness of a software visualization system, including algorithm animation systems, is currently a largely subjective and non-systematic process. Our approach employed to address this problem was to first create a software visualization ontology schema whose classes and properties consist of a reconciliation of existing taxonomies plus other features. This ontology was then employed in an interactive online tool called VisIOn [67] that allows anyone with Internet access to classify software visualization systems in a very detailed and impartial manner. As we and others continue to conduct empirical studies that investigate the contributions of individual features to the overall effectiveness of different types of software visualization systems, the entry of the results of these studies into VisIOn can then be used to provide an objective rating for each system.

Determining the pedagogical contribution and value of every feature is an enormous task that would take years for one researcher or group to accurately accomplish, since software visualizations contain hundreds of features that should be considered. Proper investigation will require that each feature, as well as combinations of features, be implemented into a software package and then studied with human subjects. In this research, we have started this process by describing and implementing a framework that includes software packages, experimental designs, techniques, and analysis, as well as examples of how to recognize the implications these results may have in the design and evaluation of effective algorithm animations.

1.4 Contributions

This section describes my contributions to this overall research project and the contribution of this research to the software visualization and computer science education communities. The research discussed in this document is a component of a larger project[1] that involved other graduate students from The University of Georgia and Georgia Institute of Technology (Georgia Tech). My involvement included:

i) Development of applications – Interactive Visualization Ontology (VisIOn) [67], Just Noticeable Difference (JND), and several VizEval modules (FileCreator, FolderFileChecker, TestFileSwitcher, and PreviewFrame) [65].

[1]*This material is based upon work supported by the National Science Foundation under Grant No. IIS-0308063. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).*

ii) Extending and maintaining software packages - System to Study the Effectiveness of Animation (SSEA) [42, 63] and the TestTaker module of VizEval [70].

iii) Experiment set-up, design, execution and analysis – Effect of flash cue on detection and localization [19], Cueing and motion in AA comprehension [64], Interactive questioning in AA comprehension, part I [66], Interactive questioning in AA comprehension, part II, Eye-tracking study, part I [40].

iv) Supervising and assisting Master's students on research ideas, software design and implementation, and preparing for and conducting empirical studies – Matthew Ross [70], Sujith Thomas [84], Bina Reed [63], Shradha Kaldate [40], Manish Agarwal [1], and Joseph Hohenstern.

The confusion and frustration stemming from the mixed results of previous studies have contributed to the sparse use of SVs in educational settings and have led some SV researchers to shift focus to other areas. However, many computer scientists within and outside of the SV community still believe that these tools, when designed properly, stand to greatly benefit those who use them. This research:

i) initiates a comprehensive listing of features of AAs and AA systems and introduces a framework for testing the effectiveness of each,

ii) takes advantage of the body of knowledge already established in perceptual psychology and cognitive psychology,

iii) offers an explanation of the inevitable variability present in studies involving human subjects through the investigation of the effects of individual differences on comprehending AAs, and

iv) classifies and makes widely and easily available SV systems, in a way never done before, with VisIOn.

Results of studies in this work have found that some perceptual/attentional features, such as cueing, increase performance on some tasks (localization) while harming performance on others (detection), and vice versa. We have also found that features such as motion, cueing, and interactive questioning may enhance attention and perception, but this enhancement does not necessarily translate into improved comprehension of an algorithm. These types of results aid in informing the design and evaluation of effective SVs and SV tools.

1.5 Overview

Chapter 2 provides background information about algorithm animations, related empirical studies, and psychology terms used to describe various AA features. Chapter 3 describes the purpose and architecture of the three software packages developed to create, conduct, and analyze experiments, and Chapters 4 and 5 describe the design, procedure, and results of the six studies conducted thus far. Chapter 6 explains the means by which we classify and propose to objectively evaluate SV systems through the creation and use of VisIOn (An Interactive Visualization Ontology). A summative analysis of the studies, a conclusion, and ideas for future work are presented in Chapter 7. Figure 1.1 provides a visual overview of this research.

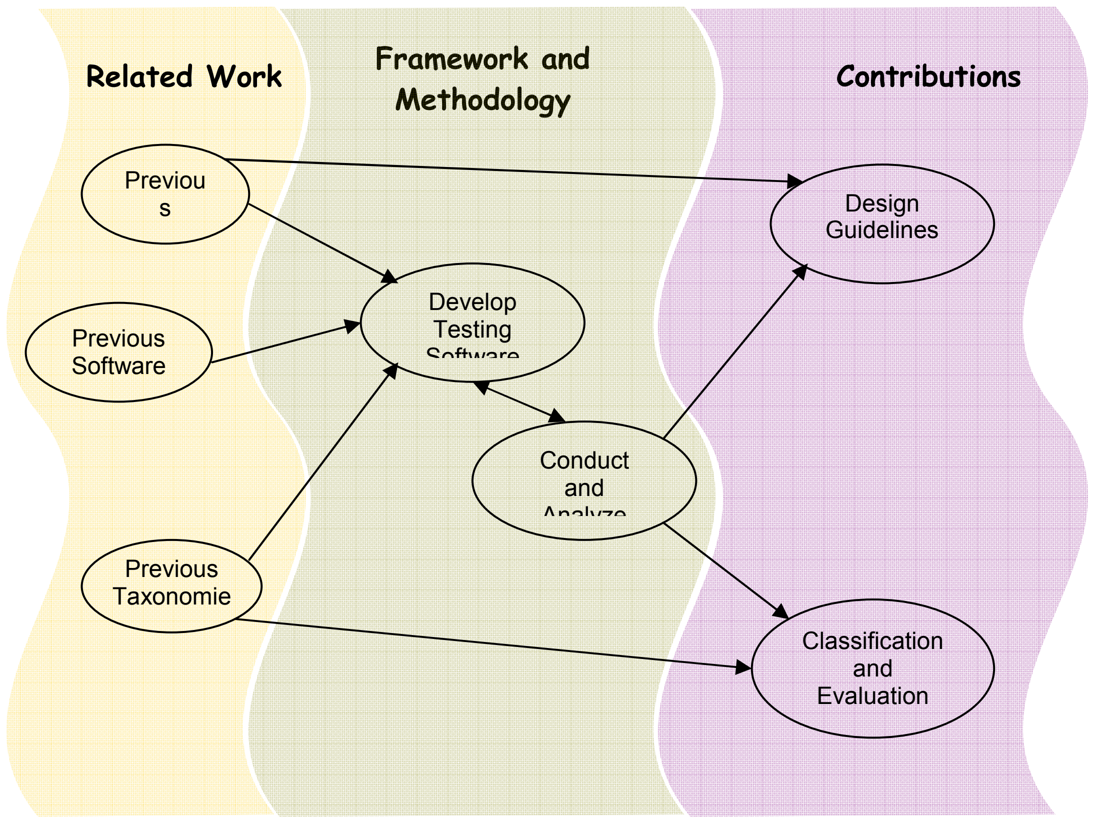


Figure 1.1: Visual overview of our research procedures and contributions.

CHAPTER 2

REVIEW OF THE LITERATURE

2.1 Algorithm Animation Background

The video entitled "Sorting Out Sorting" created by Ron Baecker [4] in 1981 is usually noted as the first algorithm animation. It was created to teach students the step-by-step operations of three different algorithms as well as to demonstrate the run-time performance of each. Shortly afterwards, others [10, 11, 76, 78] developed software systems to automate the creation of algorithm animations with the capabilities of sound, color, multiple views and multiple dimensions. More recent AA systems have been designed with the intention of students using them to easily create their own animations [79].

In the early to mid 1990's, some researchers began to test whether these animations actually added the expected educational benefits for which they were designed. Some studies found that students who used algorithm visualizations significantly outperformed those who used an alternate method to learn the algorithm [13, 44] while other studies found no difference between the performance of the two groups [26, 44]. The conflicting and unexpected results of some of the initial studies sparked an interest in determining what factors contribute to the effectiveness of an algorithm animation or system.

A number of papers have been published addressing the usefulness of AAs evaluated through anecdotal [10, 79] and empirical studies [37, 39], as well as factors to consider when designing an AA for pedagogical purposes [55, 71]. These studies have ranged from comparing the use of an AA to a non-visualization method [3, 30], to having students build their own AA [35, 79], to comparing features of one AA against those in another [44, 73]. Please see Appendix E for a "Quick Glance of Algorithm Animation Studies by Feature". My research focuses on this last approach, comparing features of one algorithm animation to another, and takes into account individual differences of participants in addition to findings and concepts from perceptual and cognitive psychology. Little related work exists for this approach.

2.2 Visualization and Psychology

Designing and evaluating algorithm animations to make them more effective for users fall within the human factors and human-computer interaction domains. *Human factors psychology* is an "interdisciplinary field which discovers and applies information about human behavior, abilities, limitations and other characteristics to the design and evaluation of products, systems, jobs, tools, and environments for enhancing productive, safe, and comfortable human use" [62]. It involves the application of perceptual and cognitive processes to improve the usability of applications and the performance of their users. *Human-Computer Interaction* (HCI) involves the study and practice of usability. It includes creating technology that "people will want to use, will be able to use, and will find effective when used" [15]. Preece et al. [59] believe that the main objective of HCI has been to "understand and represent how humans interact with computers in terms of how knowledge is transmitted between the two."

Consideration of perceptual, attentional, and cognitive concepts is important in the development of effective visualizations. *Cognitive psychology* is the study of internal mental processes such as problem solving, decision making, language, and short- and long-term memory [24]. Knowledge gained through perception, reasoning, or intuition is *cognition* [20]. The AAs and systems that we study were designed for users to gain knowledge about the depicted algorithms. We describe cognitive features of AA systems, such as interactive questions, to be those that require a user to solve problems or think about the implementation of the algorithm she is viewing. Since knowledge can be gained through perception, we investigate features of AAs that may affect how a viewer perceives the animation.

Perception is the process of acquiring, interpreting, and organizing sensory information [24]. Factors that affect perceptual performance include arousal, fatigue, mental load, monotony, boredom, sensory deprivation, sleep deprivation, anxiety, fear, isolation, and aging [48]. Currently, AAs are perceived using vision and some have tapped into the use of auditory senses. Perceptual characteristics of AAs include components of the layout such as data set size and the use of color and sound.

Attention is defined as the cognitive process whereby a person concentrates on some features of the environment to the relative exclusion of others [24, 46]. "Learning is most efficient when a person is paying attention" [50]. Attention can be allocated voluntarily or involuntarily. As described by Prinzmetal et al. [61], voluntary attention, also referred to as goal-directed or endogenous attention, occurs when observers "allocate attention to the spatial location that may contain information that is important to immediate task goals". Involuntary attention, also referred to as stimulus-driven or exogenous attention, occurs when an observer's attention is captured by a stimulus event "even when the stimulus event is unrelated to the current goal-directed activity". Different types of cues, questions, and motion can be used within algorithm animations to guide a viewer's attention to important actions occurring on specific portions of the screen.

Cognitive load theory addresses the limited capacity of the independent channels in the human information-processing system [49, 58]. The theory is based on a cognitive design that involves a limited working memory that interacts with a relatively unlimited long-term memory. The theory explains how only a restricted amount of cognitive processing can occur in the visual/spatial channel or the auditory/verbal channel at any given time.

Researchers in various fields have performed studies to focus on different factors that they hypothesize will have an impact on the effectiveness of a visualization or visualization tool used within their respective environments. In general, features pertaining to the design of the display and to interaction with the tool have been evaluated [34, 44, 72], and an overlap exists in the types of evaluations conducted. For example, the idea of creating appropriate visual metaphors discussed by Rhyne et al. [68] for use in bioinformatics visualizations coincides with the idea of increasing the use of the human visual system by using graphics that are easily perceived [74]. Such graphical displays are believed to reduce the level of cognitive effort needed to interpret the represented concept, thus creating a cognitive economy [85] in which the visualization is easy to understand but not oversimplified [86].

2.3 Studies of Individual Features

When designing visualizations or visualization tools, "information needs to be presented in a way that makes it unambiguous to perceive and understand" [59]. One challenge is to identify perceptual factors of a visualization or visualization tool that could be used to help reduce the viewer's cognitive load and then to properly isolate, analyze, and apply results to a "real-world" application. Studies on perceptual factors such as color, sound, and touch have been successful in identifying proper usage in creating tools for specific domains [33, 34, 82]. For instance, color has been found to be most useful for identification tasks and as a form of redundant coding [59].

Similar to research done by Healy et al. [33, 34] for information visualizations, our work involves conducting low-level, perceptual studies and using the results to inform the design of algorithm animations. However, our work differs in that we have taken our analysis a step further to empirically evaluate the effectiveness of the individual features on the performance of the higher-level, cognitive task of comprehending the algorithm.

The term "de-feature" is used by Morse et al. [51] to describe how they approached their studies of visualizations used in the information retrieval domain. They point out that usability studies tend to focus on visualization systems as a whole and not the visualization itself. They were able to gain insight into the effect of various display techniques on a user's ability to accurately perform tasks and on the time to completion. For complex tasks, they also found a correlation between participant preference and performance and felt that the "subjects like to use things that make them successful".

2.4 Algorithm Animation System Design

Röβling et al. [71] feel that effective learning from algorithm visualizations (AVs) probably will not be achieved purely with "bigger and better graphics", and they have proposed nine pedagogical requirements for developing AVs that address features such as the architecture, the display design, and the interaction provided to and encouraged of the user. Naps et al. [55] provide a similar list of best practices for designing educational visualizations and argue that "visualization technology, no matter how well it is designed, is of little educational value unless it engages learners". However, many of these suggestions have not been proven to be beneficial, and in some cases empirical studies have produced results that contradict the usefulness of some of these recommended features. For instance, the use of interactive questioning and multiple views have been recommended by both groups, but findings have shown that these features slightly harm performance [38, 66] and increase the time spent using the system without increasing learning [73], respectively.

An agenda for evaluating graphical representations was described by Scaife and Rogers in which consideration is given to external representations, internal representations, and the interaction between the two [74]. They provide a survey of related studies and identify useful findings but also point out assumptions and/or lack of proof of certain assertions made by other researchers. Some of these most popular claims are:

- ► static visualizations are better than textual descriptions.
- ► animated visualizations are more effective than static ones.
- ► virtual reality is better than animation.
- ► three-dimensional diagrams are better than two-dimensional.
- ► solid modeling is better than wire-frame modeling.
- ► color is better in visualizations than black and white.
- ► interactive graphics are better than non-interactive graphics.

They state that these generalizations have been made without adequate proof of what is cognitively gained by graphical representations that are more explicit, interactive, and dynamic.

Tudoreanu [85] explains the concept of *cognitive economy* as it pertains to the algorithm animation domain. It “seeks to minimize the load of the cognitive system” by reducing the complexity of a visualization while maximizing the amount of important information shown. Through empirical studies, he identifies the management of a user’s cognitive load as being a key factor in determining the effectiveness of a visualization. He states that visualization environments that are too complex and increase the user’s cognitive load are no more beneficial than viewing no visualization at all.

The design of an animation and proper interaction are key factors contributing to the effectiveness of pedagogical tools, specifically algorithm animations. Features that can be classified as either animation or interaction design have been evaluated, but in a somewhat random or non-systematic manner. The remainder of this section discusses these studies as they relate to design and interaction features of algorithm animations.

2.4.1 Animation Design

Components of an animation include graphical objects, color, sound, motion, textual labels, captions, a code view, and visual cues. At times, these type of features, referred to as "representational characteristics"[36], have been discredited as having little pedagogical benefit [36, 55, 71] because only a few studies investigating these features have led to statistically significant results [44, 73].

Lawrence studied user preferences and the effects of labels on objects, varying data set sizes, and choice and orientation of graphical objects [44]. While she found the variations in these attributes did not increase learning, the results have been useful in showing that these features do not decrease learning and that viewers' preferences do not necessarily make a difference in algorithm comprehension [3, 44]. Meanwhile, other studies have shown that features of the animation design contributed to a significant increase in learning. For instance, allowing a user absolute control over the speed [73], including textual and visual cues [44], and reducing the number of colors used in the display [44] have all been found to enhance the pedagogical value of algorithm animations. The usefulness of these features was determined by participants' performance scores on assessments of their comprehension of the animated algorithm presented during a session. Our research investigates the effect of these types of features on comprehension as well as the viewer's ability to perceive the animation.

2.4.2 Interaction Design

Grissom et al. [27] established an engagement taxonomy that categorizes six levels of interaction of students with visualization tools. In order of increasing engagement, the categories are:

1. No viewing,
2. Viewing,
3. Responding,
4. Changing,
5. Constructing, and
6. Presenting.

Even though the categories can overlap, the hypothesis is that the greater the level of engagement, the greater the learning benefit. This taxonomy coincides with research published in 1954 [18] that has since been described as the "Learning Pyramid" [56] represented in Figure 2.1.

The "Learning Pyramid" illustrates an increase in the retention of information as more interactive learning methods are utilized. Similar to the learner engagement taxonomy, students are believed to retain more knowledge as they perform tasks that more actively involve them in the learning process. Our studies on interactive questioning techniques compare viewing to responding as well as types of questions within the responding category. The studies discussed in this section have investigated differences in performance in relation to the levels of this taxonomy.

Several successful attempts have been made to show that students who actively interact with a visualization technology enjoy an enhanced learning experience over those who only attend a lecture [13, 27, 30] or read a textual description [30] with static images. However, no significant difference in performance was found between the two groups in similar studies conducted by the same researchers [30] when participants in the non-visualization group were given a "carefully designed" handout with exercises.

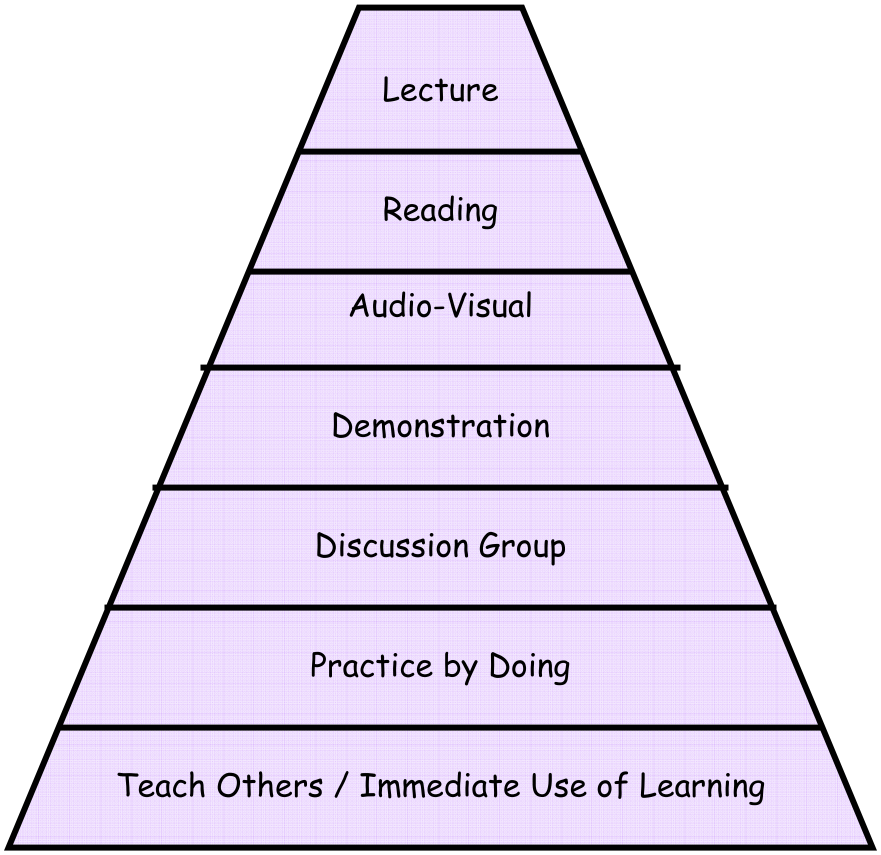


Figure 2.1: A representation of the "Learning Pyramid" [56].

In some studies in which a performance increase was found, the specific contributing factors could not be discerned. For example, a study investigating the effects of prediction and animation on algorithm comprehension found that subjects who were asked to make predictions while viewing an animation performed significantly better on post-tests than the other three groups (prediction only, animation only, and neither). However, it was not clear whether prediction or animation alone were significant factors, or if it was only the combination of the two that had a significant effect.

The goal of studies presented in [39] was to determine whether predictive interaction has a positive impact on learning. After conducting two studies in which students interacted with the Interactive Data Structure Visualizations (IDSV) courseware, Jarc et al. concluded that predictive interaction was not beneficial in general and that it may be most useful for difficult concepts and dynamic displays.

Hansen et al. [30] performed a series of experiments in which they compared the use of their fully functional Hypermedia Algorithm Visualization (HalVis) system against text-only learning, lecture, XTango (another visualization system) [77], and "reduced" versions of HalVis in which some views or features were removed. Of their eight experiments, two are related to our current research. One experiment involved participants passively viewing an animation presented by XTango compared to those who were interactively taught the same algorithm using HalVis. Even though the HalVis group significantly outperformed the XTango group on a post-test, HalVis offers additional views (i.e. a conceptual view that shows a real-world analogy to the algorithm) and features, such as probes, than those provided by the chosen XTango animation, and none of these characteristics can be singled out as the reason for the better performance of the HalVis group [30]. Therefore, other experiments were conducted in which a full-version HalVis was compared to versions without certain views or features. As a result of these experiments, they found that participants who were required to answer pop-up questions performed better, though not significantly, than those who were not [30].

The recommendation to allow students to enter their own input set into an algorithm animation [55, 71] falls under the "changing" category of the student engagement taxonomy

[27]. Lawrence et al. conducted two studies [44, 45] that found a significant improvement in performance for students who entered data sets compared to those who did not. In contrast, Saraiya et al. [73] found that students who entered their own data performed slightly worse than those who were given good examples. In the Lawrence et al. studies, the participants who entered their own data sets were exposed to the algorithm either through written text or lecture before attempting to create examples, whereas it is not clear if the students in the last study were given any introductory material about the algorithm and were likely to have entered random, meaningless data sets.

Constructing an algorithm visualization, the fifth category in the student engagement taxonomy, can be accomplished through hand construction or direct generation in which the learner maps the algorithm to a visualization through some type of code augmentation or script. In some studies, students who created algorithm animations were reported to have improved performance [35] and to have found the course more enjoyable because it was more engaging [79] while another study found no effect [36].

2.5 Summary

Hundreds of algorithm animations and systems exist including those that can run over the internet [39], as a standalone application [79], or both [31]. This related work section has highlighted key eras in the evolution of algorithm animations. When AAs were first brought into existence in the early 1980's [5], the main focus was on developing a visualization tool that could dynamically show students the execution of an algorithm. In the early to mid 1990's, some researchers became concerned with classifying software visualizations and testing whether they actually added the expected educational benefits for which they were designed. The varied and unexpected outcomes of some of the initial studies [44] resulted in additional studies and recommendations of factors believed to contribute to the effectiveness of an AA or AA system [55, 73].

This research investigates the effectiveness of algorithm animations and systems from both a design and an evaluation perspective. Assumptions about the usefulness of various features and interaction have been turned into hypotheses and were empirically evaluated. Since some AAs have proven to be effective educational tools while others have not, the implementation of certain features within the various AAs must contribute to these differences in effectiveness. Therefore, we compare AAs against one another by contrasting feature(s) between the two and conducting user studies with student participants. Some of the variation found in these studies on the effectiveness of AAs on comprehension may also be due to individual differences amongst students. We have taken this into consideration and started collecting and analyzing assessments designed to differentiate students based upon the type of learner they identify themselves to be and scores on various ability tests. The results of these studies will aid in designing effective algorithm animations and in objectively evaluating those that already exist.

CHAPTER 3

FRAMEWORK AND METHODOLOGY

3.1 Introduction

The approach undertaken in this work to design effective software visualizations bears some similarity to research done in the information retrieval [51] and information visualization [32] domains. Our method involves de-featuring visualizations and systems to first identify individual characteristics that are commonly used in existing AAs or that were found to be of interest. In our work, these features are investigated at a perceptual level, meaning they are presented in a simple, context-free display in which studies are conducted on the viewer's ability to detect and localize changes. These features are then integrated back into an algorithm animation and higher-level studies are performed to determine if the features found to enhance perception will in turn enhance cognition or the viewer's ability to comprehend the depicted algorithm.

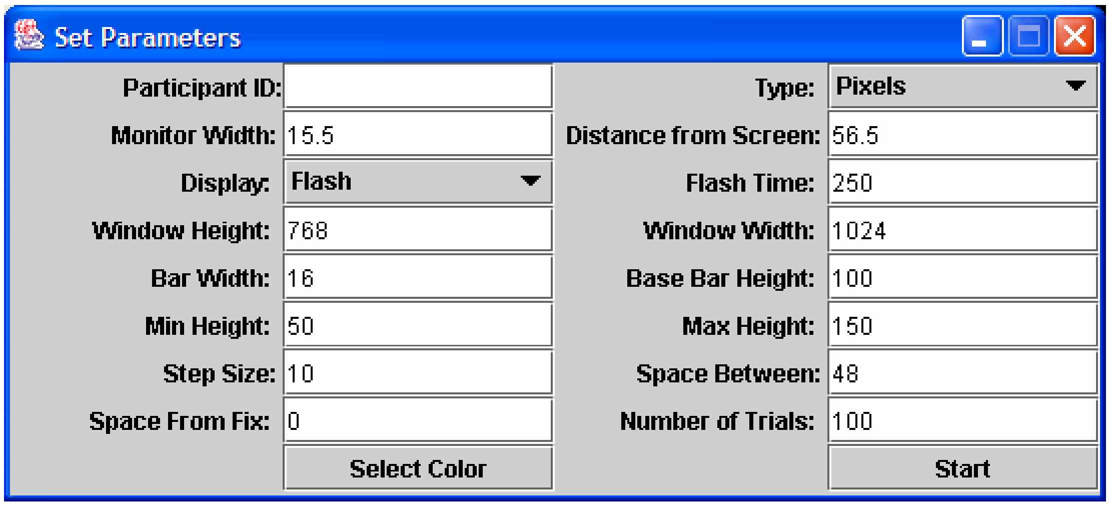
To accomplish these tasks, several software packages have been developed to automate and simplify the creation, execution, and analysis of experiments designed to test features of algorithm animations and systems. This chapter describes the overall purpose and architecture of the software and provides an overview of how an experimenter may use the various graphical user interfaces to accomplish their tasks. Specific scenarios of the usage of these software packages are detailed in Chapters 4 and 5 with descriptions of how the software has been used to create and conduct empirical studies.

3.2 Just Noticeable Difference (JND) Software

Just Noticeable Difference, JND, is a testing application developed to conduct two-alternative, forced-choice, discrimination threshold experiments. Two-alternative, forced-choice experiments require participants to select one of two pre-determined options even if they do not believe either of them to be true. Discrimination threshold experiments ask participants “to tell apart, or discriminate, two things that differ by only a slight increment” [43]. The results can then be analyzed using Weber’s Law which states that the “smallest difference in a specified modality of sensory input that is detectable by a human being” [24], or JND, is a constant ratio of the change in a stimulus and the original stimulus [7]. For studies conducted with this software, these terms mean that a viewer has to visually determine the taller of two bars and that the results can then be used to calculate a (just) noticeable difference for bar heights.

The reason JND was developed was to conduct a preliminary study to determine the minimum difference between heights of bars that can be detected at a 95% confidence level. In the algorithm animations studied in this research, vertical rectangular “bars” are used to represent data elements. The bar’s location indicates the index within an array and the height indicates the value. To ensure that viewers are able to perceive the differences in values of elements depicted by the bars, JND has been used to conduct an experiment that provides the minimum step size, or difference between bar heights, for displays in our other studies.

At the beginning of each participant session, the administrator may enter or modify data about the participant, the monitor, and the display (see Figure 3.1). The ability to specify the variables displayed on this “Set Parameters” screen at runtime is provided for the perceptual psychologists who design and conduct these low-level studies. The “Monitor Width” and the “Distance from Screen” must be entered in inches. The “Display” of the bars may be a “Flash”, shown for a specified amount of time and then hidden, or “Fixed”, shown until the participant selects an answer. If “Flash” is chosen, then the administrator must enter the length of the flash in milliseconds in the “Flash Time” textbox. The “Number of Trials” parameter will be the number of displays and questions shown to the viewer.



Participant ID:		Type:	Pixels
Monitor Width:	15.5	Distance from Screen:	56.5
Display:	Flash	Flash Time:	250
Window Height:	768	Window Width:	1024
Bar Width:	16	Base Bar Height:	100
Min Height:	50	Max Height:	150
Step Size:	10	Space Between:	48
Space From Fix:	0	Number of Trials:	100

Figure 3.1. Screenshot of the set-up window of JND. The experiment administrator can overwrite the default values for each parameter.

All other parameters can be entered in “Pixels” or “Degrees”, whichever is selected by changing the value of “Type”. The degrees are computed based on the monitor width and the participant’s distance from the screen. The angle of view (see Figure 3.2) is calculated and then the ratio of pixels per 1^{o} is used as a conversion factor. As the administrator toggles between the “Pixels” and “Degrees” options, the values are automatically converted from one measurement to the other.

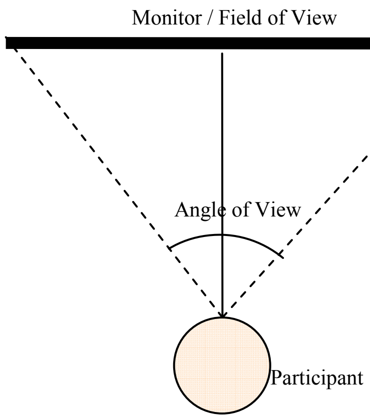


Figure 3.2. A depiction of an overhead view of a participant during an experiment session.

The “Select Color” button opens the screen shown in Figure 3.3 and allows the administrator to select the desired color of the bars to be displayed. The values shown in Figures 3.1 and 3.3 are the default values that were researched and agreed upon by members of our research group. The use of such default values reduces the time needed to set up each experiment.

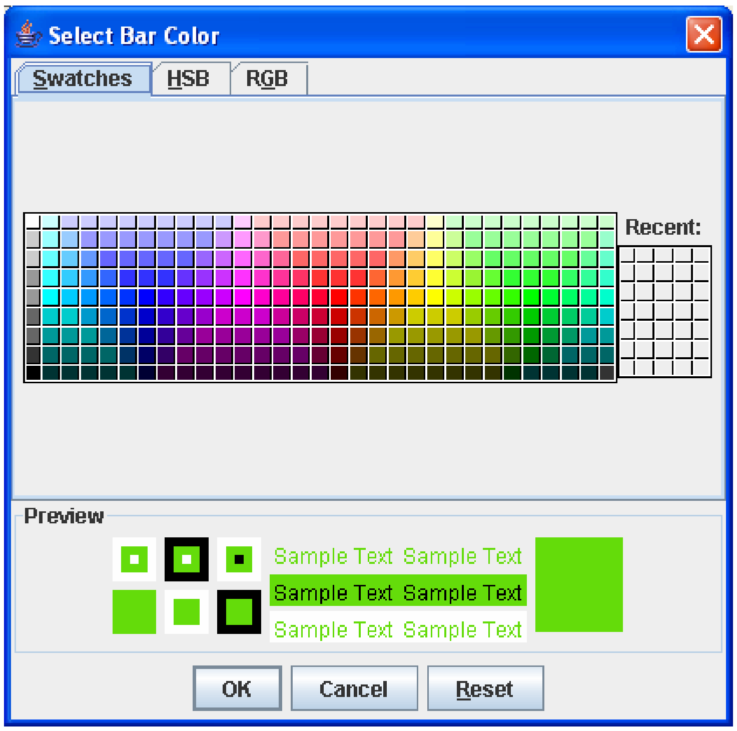


Figure 3.3. The color palette screen of JND. It is used to select the color of the bars. The default value shown is green, Red / Green / Blue (RGB): 100 / 220 / 10

Once the "Start" button on the "Set Parameters" window is pressed, a small screen (see Figure 3.4) appears requesting the participant to press any key to continue. The purpose of this prompt is to guide the viewer's focus to the same fixation point (the black dot) at the beginning of each trial.

The viewer is then shown two vertical, rectangular bars, as illustrated in Figure 3.4, and must decide which bar is taller. Depending upon the option selected by the experiment administrator, the bars may stay on the screen or they may appear for a specified number of milliseconds. Since the user must judge which of the bars is taller and then select either the left or right bar, this is called a two-alternative forced-choice judgment.

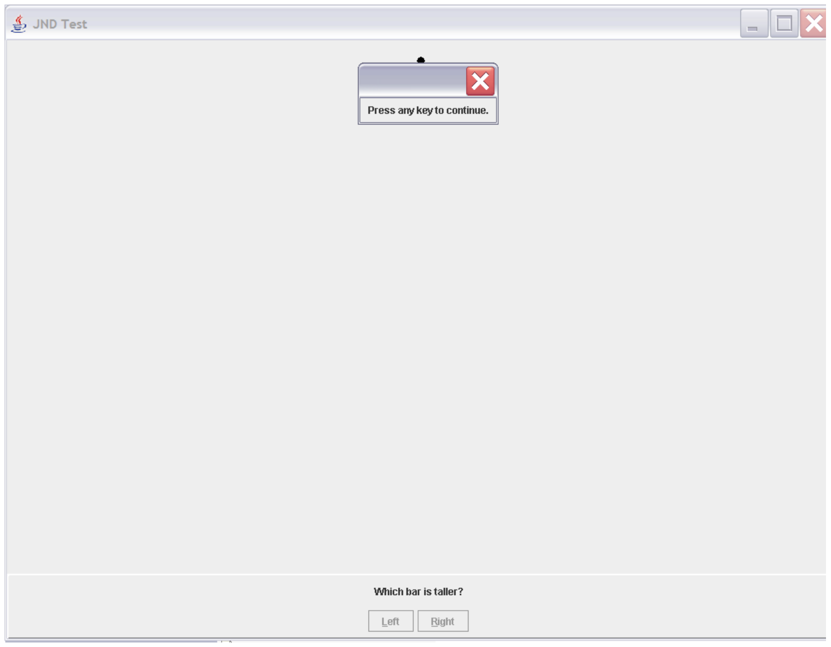


Figure 3.4. Prompt to guide viewer's focus to the same location at the beginning of each trial.

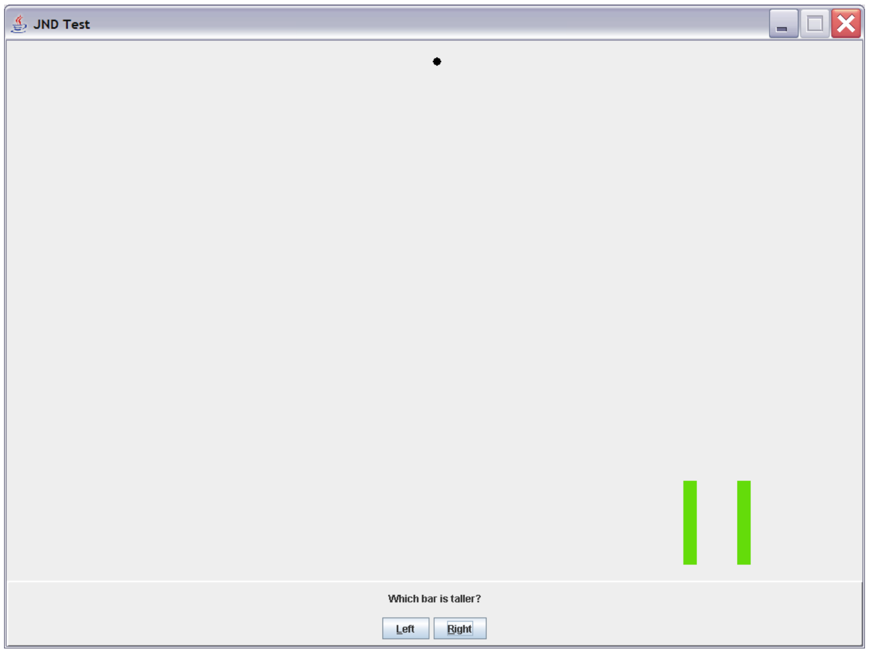
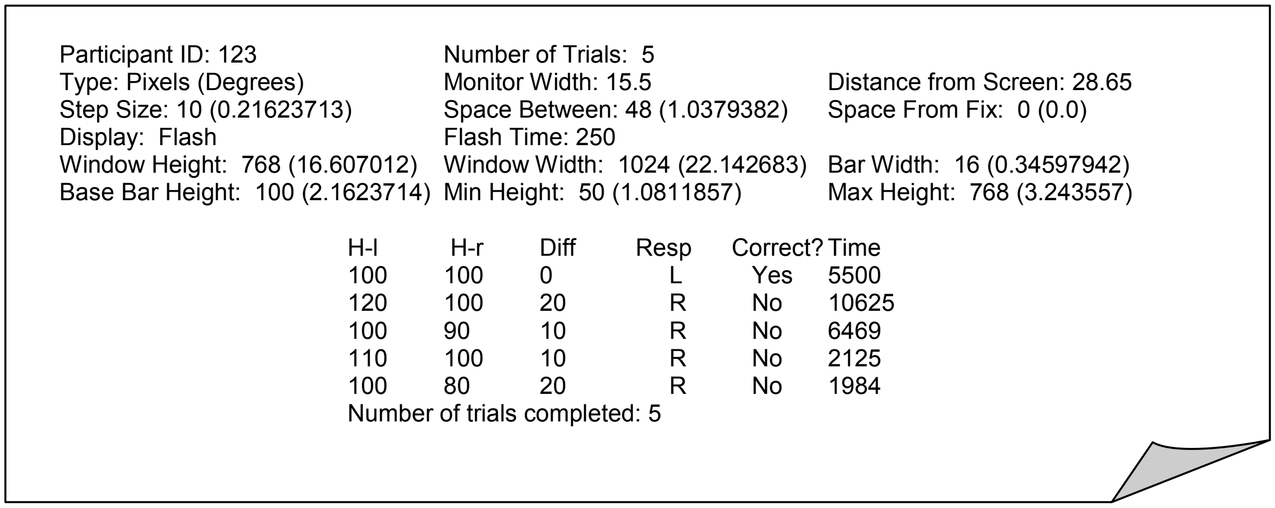


Figure 3.5. Sample display and question.

After the participant has completed all trials, a logfile is written with information about the session (see logfile__12.11.2006_13.35.58.txt in Figure 3.6). It is a simple text file, and its name consists of the date (_mm.dd.yyyy) and time (_hh.mm.ss) the session began.


```
Participant ID: 123                  Number of Trials:  5
Type: Pixels (Degrees)               Monitor Width: 15.5                   Distance from Screen: 28.65
Step Size: 10 (0.21623713)           Space Between: 48 (1.0379382)         Space From Fix:  0 (0.0)
Display:  Flash                      Flash Time: 250
Window Height:  768 (16.607012)      Window Width:  1024 (22.142683)       Bar Width:  16 (0.34597942)
Base Bar Height:  100 (2.1623714)    Min Height:  50 (1.0811857)           Max Height:  768 (3.243557)

                                 H-l      H-r      Diff       Resp    Correct? Time
                                 100      100      0            L       Yes    5500
                                 120      100      20           R       No     10625
                                 100      90       10           R       No     6469
                                 110      100      10           R       No     2125
                                 100      80       20           R       No     1984
                                 Number of trials completed: 5
```
Figure 3.6. Sample logfile produced by one session of an experiment using JND.

The parameter values entered on the set-up screen are printed at the beginning of the file. Then, for each trial, the heights of both the left and right bars, the difference between the two, the user-entered response, the correctness of the response, and the response time are listed. The data is entered in a tab-delimited format that can easily be copied into Microsoft Excel or read by a statistical software package for analysis. The "Noticeable Difference Experiment", described in Section 4.2, was conducted using JND.

3.3 VizEval Package Suite

The VizEval Suite [65] is an environment designed to support experimentation with the effect of various attributes of algorithm animation on the user's ability to perceive and attend to objects and events occurring in software visualizations. This is a software package that consists of several components (see Figure 3.7): *FileCreator*, *TestCreator* [84], *TestTaker* [70], and utility programs. Each component serves a role in simplifying the process of creating, conducting, and analyzing studies of perceptual and attentional features of software visualizations. The VizEval Suite allows the experiment administrator to develop an experiment (including specification of all of the graphics, animations and questions), to deploy that experiment, and to collect and organize the output. Automation is important because of the size and complexity of these perceptual studies, which may consist of hundreds of trials and complex orderings within a trial or across test participants.

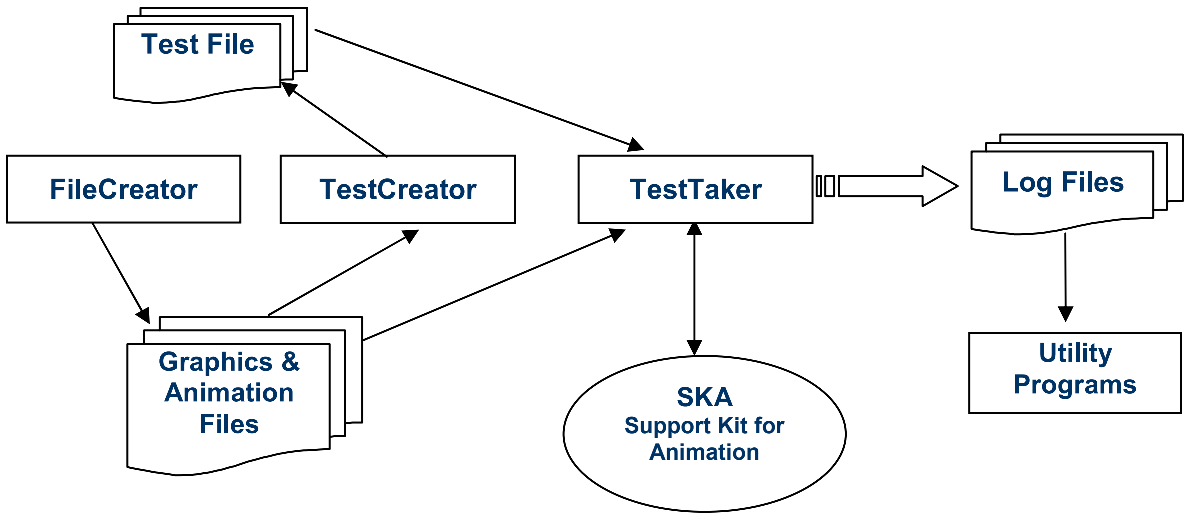


Figure 3.7. The VizEval Architecture.

Each experiment consists of a number of blocks. Each block contains some number of trials. In each trial, the participant views a short animation and is then asked a series of questions about what she saw and understood.

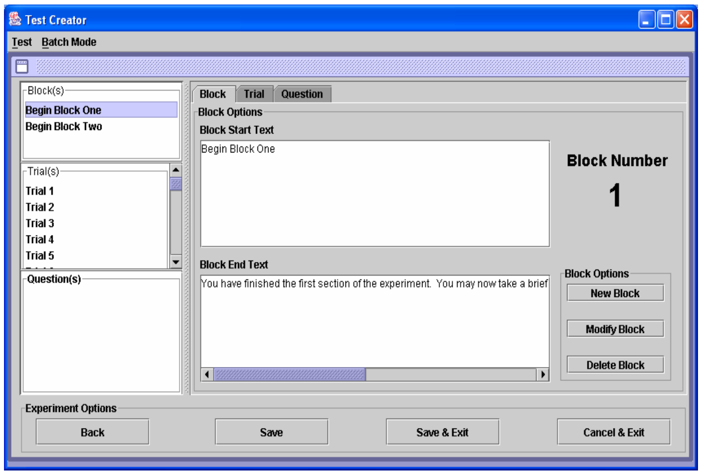
The experiment administrator uses *TestCreator* to specify the components of the experiment. For large experiments involving a substantial number of graphics and animation files, the *FileCreator* module may be used to automate their generation. The *TestTaker* module executes the experiments: displays visualizations, presents questions, and records user responses, timing and other information in a log file.

SKA [29] is a combination of a visual data structure library, a visual data structure diagram manipulation environment, and an algorithm animation system, all designed for use in an instructional setting. In the context of the VizEval suite, SKA serves as the graphics and animation engine.

Graphical objects and their animations are specified in graphics and animation files, respectively. These are simple text files that are processed by SKA at run-time. While such files may be created manually using a text editor, it is desirable in the case of large experiments to use *FileCreator* to automate this process. Researchers may allow the module to automatically generate a full set of graphical objects of all set sizes and of randomized heights, or may specify particular set sizes, heights, or other attributes. The administrator can also create animation files. Either an individual file or a set of files can be generated, using either experimenter-specified or random values.

TestCreator [84] facilitates the design and generation of experiment test files. It leads the experiment designer step-by-step through the process of specifying each block, the trials within each block, and the graphics, animations, and questions associated with each trial (see Figure 3.8). *TestCreator* features support for five different types of questions: mouse (requires user interaction with a mouse), keyboard (requires interaction through the keyboard), multiple choice, N-Point (e.g. Likert Scale), and Yes/No (True/False). Additional customized question types may be created by extending the class "Question".

The experiment file shown in Figure 3.9 is generated by *TestCreator* and contains all information needed to run the experiment, including user instructions, questions, start method (enter key, mouse click, space bar), attractor (countdown timer, etc.), graphics and animation files to be used, as well as various timing and flow of control parameters.

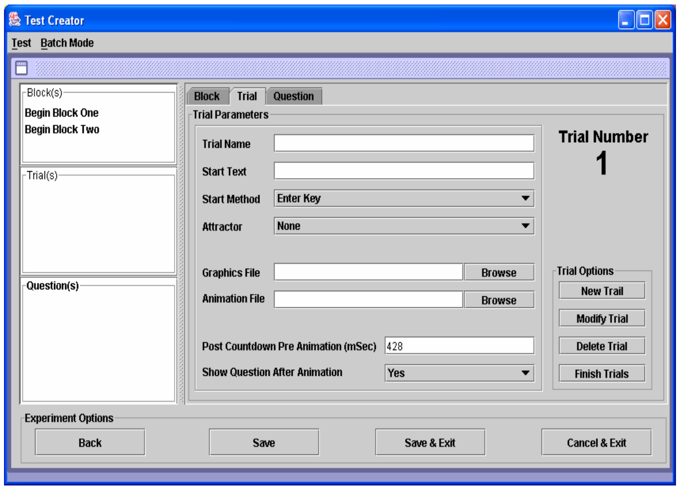


Block(s): Begin Block One, Begin Block Two

Trial(s): Trial 1, Trial 2, Trial 3, Trial 4, Trial 5

Block Start Text: Begin Block One

Block End Text: You have finished the first section of the experiment. You may now take a brief



Block(s): Begin Block One, Begin Block Two

Trial Name:

Start Text:

Start Method: Enter Key

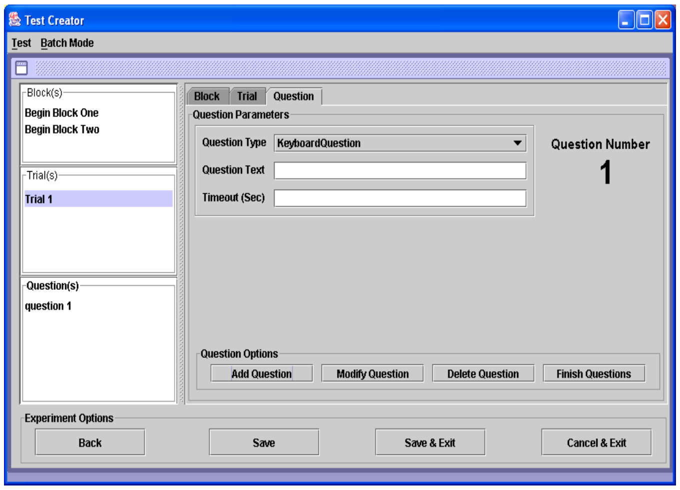
Attractor: None

Graphics File:

Animation File:

Post Countdown Pre Animation (mSec): 428

Show Question After Animation: Yes



Block(s): Begin Block One, Begin Block Two

Trial(s): Trial 1

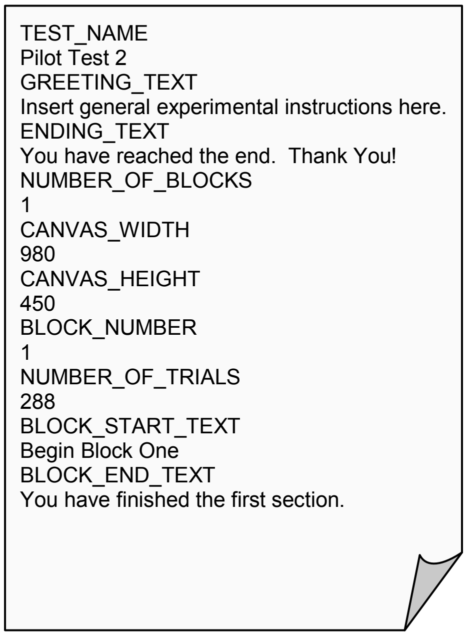
Question(s): question 1

Question Type: KeyboardQuestion

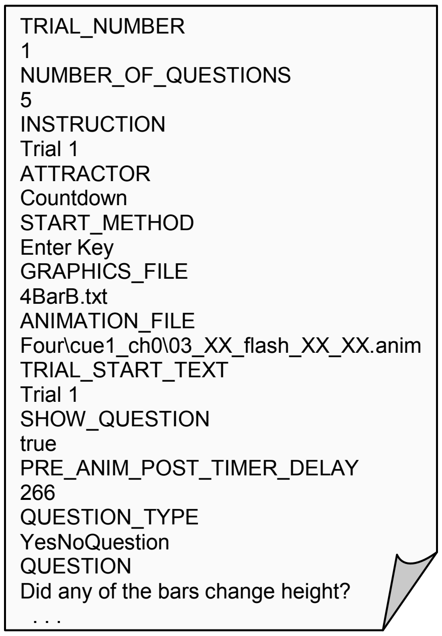
Question Text:

Timeout (Sec):

Figure 3.8: Screen shots of *TestCreator*. The "Block" (top), "Trial" (bottom left), and "Question" (bottom right) windows are each used to specify information about the experiment.



```
TEST_NAME
Pilot Test 2
GREETING_TEXT
Insert general experimental instructions here.
ENDING_TEXT
You have reached the end.  Thank You!
NUMBER_OF_BLOCKS
1
CANVAS_WIDTH
980
CANVAS_HEIGHT
450
BLOCK_NUMBER
1
NUMBER_OF_TRIALS
288
BLOCK_START_TEXT
Begin Block One
BLOCK_END_TEXT
You have finished the first section.
```



```
TRIAL_NUMBER
1
NUMBER_OF_QUESTIONS
5
INSTRUCTION
Trial 1
ATTRACTOR
Countdown
START_METHOD
Enter Key
GRAPHICS_FILE
4BarB.txt
ANIMATION_FILE
Four\cue1_ch0\03_XX_flash_XX_XX.anim
TRIAL_START_TEXT
Trial 1
SHOW_QUESTION
true
PRE_ANIM_POST_TIMER_DELAY
266
QUESTION_TYPE
YesNoQuestion
QUESTION
Did any of the bars change height?
  . . .
```

Figure 3.9: Sample test file created by *TestCreator* and read by *TestTaker.*

TestTaker [70] is the execution environment for the experiments. It keeps track of the user data, the date and time at which the experiment was conducted, and other metrics such as height and width of the screen, distance of the eyes from the screen and directory into which the log files are written. *TestTaker* uses SKA to display the graphical objects and to execute the animation. Associated questions are then displayed. User responses and other needed information are written into a log file.

Figure 3.10 depicts a sample user session. In this case, eight bars without labels are shown. One or two bars have been cued (by flashing or changing color) and zero, one or two bars have changed height. The users are then asked a series of questions to determine if they noticed that something changed (detection) and can identify the object that changed (localization).

The sample log file in Figure 3.11 was created by *TestTaker* and displays all of the data captured during a session in plain-text. The perceptual studies that have been conducted with VizEval thus far have consisted of hundreds of trials for each participant. Although the log files can be read and manually analyzed, *LogfileParser* was created to automate this process, save hours of time, and reduce the likelihood of errors.

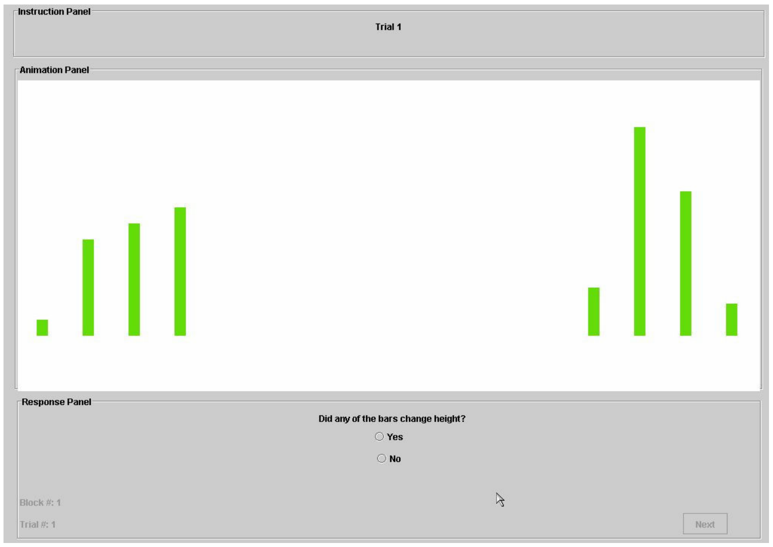
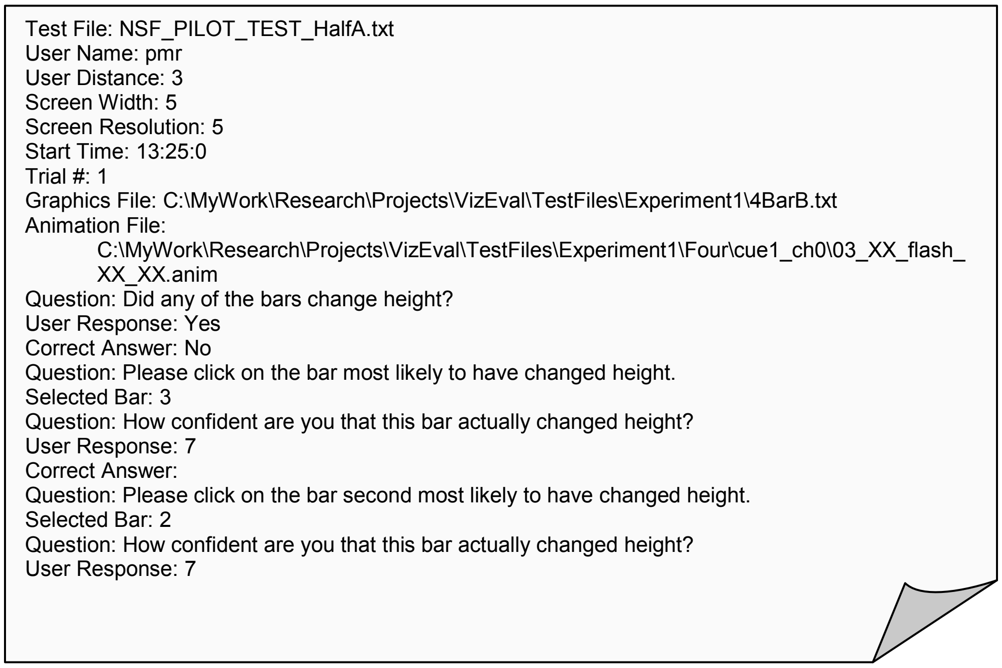


Figure 3.10: The *TestTaker* interface during a session.



```
Test File: NSF_PILOT_TEST_HalfA.txt
User Name: pmr
User Distance: 3
Screen Width: 5
Screen Resolution: 5
Start Time: 13:25:0
Trial #: 1
Graphics File: C:\MyWork\Research\Projects\VizEval\TestFiles\Experiment1\4BarB.txt
Animation File:
        C:\MyWork\Research\Projects\VizEval\TestFiles\Experiment1\Four\cue1_ch0\03_XX_flash_
        XX_XX.anim
Question: Did any of the bars change height?
User Response: Yes
Correct Answer: No
Question: Please click on the bar most likely to have changed height.
Selected Bar: 3
Question: How confident are you that this bar actually changed height?
User Response: 7
Correct Answer:
Question: Please click on the bar second most likely to have changed height.
Selected Bar: 2
Question: How confident are you that this bar actually changed height?
User Response: 7
```

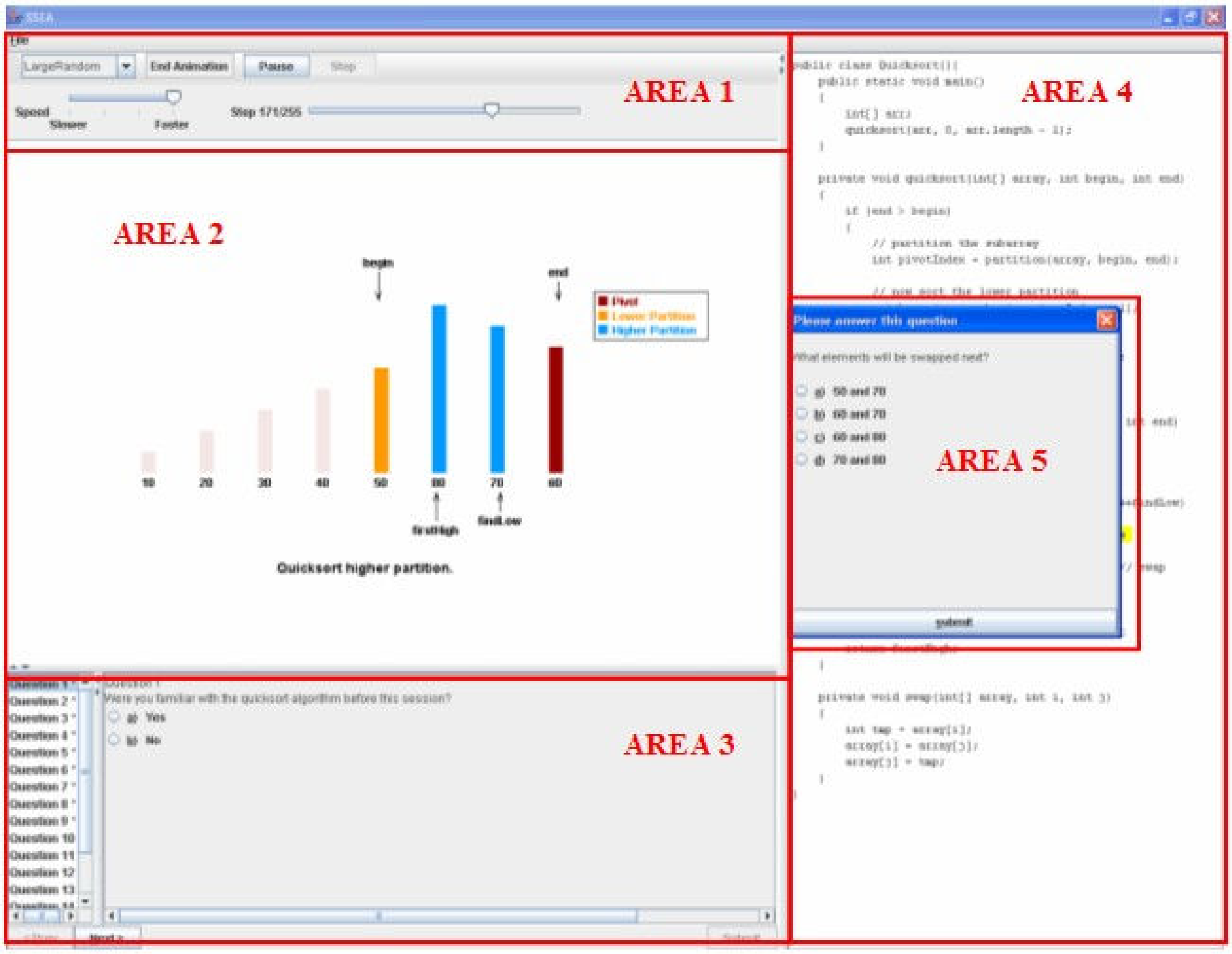
Figure 3.11: Sample log file created by *TestTaker.*

Other utility programs (*TestFileChecker*, *TestFileSwitcher*, and *FilePreview*) also were created to save time and reduce errors. Since the test file read by *TestTaker* can be edited, graphics and animation file names can be changed, and directory structures can be altered, *TestFileChecker* was created to verify that all of the files that were specified within the test file exist at the given locations. For some studies, the administrator may want all participants to view the same pairs of animations and graphics, but in a different order. Originally, this was accomplished by the administrator either re-ordering the hundreds of files by cutting-and-pasting their names into an existing test file or by creating new test files with *TestCreator* and re-entering the hundreds of trials and questions. *TestFileSwitcher* will read in a test file, randomly order the pairs of animation and graphics files, and print a new test file with the same instructions and questions as the original test file. *FilePreview* allows the administrator to quickly preview a graphics file alone or in combination with an animation without starting a session of *TestTaker*.

VizEval Suite has been and is being used to assist researchers in conducting experiments on attributes that may contribute to the effectiveness of software visualizations. Compared to manual specification, VizEval provides a substantial time saving for the perceptual psychologists who are using the software. Even though VizEval was created specifically for the types of tasks described above and in Section 4.3, it is a general application that can support experiments using other types of graphics, animations, and questions.

3.4 System to Study the Effectiveness of Animations (SSEA)

A System to Study the Effectiveness of Animations, SSEA, is a testing environment designed to study the effectiveness of dynamic software visualizations [42, 63]. It was created as a tool to investigate the effects of various design and interaction features of algorithm animations on a viewer's ability to perceive and comprehend a depicted algorithm. SSEA is capable of running an animated graphical view (see Figure 3.12, AREA 2) simultaneously with an animated textual view (see Figure 3.12, AREA 4) of an algorithm. The textual view is described as animated because the line of the source code that is currently being executed and graphically animated is highlighted. Users can select a data set, pause, rewind and control the speed of the animation (see Figure 3.12, AREA 1).



```
public class Quicksort(){
    public static void main()
    {
        int[] arr;
        quicksort(arr, 0, arr.length - 1);
    }

    private void quicksort(int[] array, int begin, int end)
    {
        if (end > begin)
        {
            // partition the subarray
            int pivotIndex = partition(array, begin, end);

            // now sort the lower partition
```

What elements will be swapped next?

- a) 50 and 70
- b) 60 and 70
- c) 60 and 80
- d) 70 and 80

```
        return firstHigh;
    }

    private void swap(int[] array, int i, int j)
    {
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }
}
```

Question 1

Were you familiar with the quicksort algorithm before this session?

- a) Yes
- b) No

Figure 3.12: Screenshot of SSEA.

An experiment can consist of questions asked via a pre-animation questionnaire or test, a post-animation test, or during the animation with pop-up questions (see Figure 3.12, AREA 5) and "traditional" questions (see Figure 3.12, AREA 3). Pop-up questions appear over the source code, pause the animation, and require a viewer to answer it before the animation will proceed. The experiment administrator can specify when a pop-up question appears by modifying the source code that implements the algorithm. The text of the question is contained in an XML file. The "traditional" questions are located at the bottom of the screen and can be answered at any time and in any order. The viewer is free to use the animation and displayed source code to assist in answering the questions.

Each action performed by users as well as their responses to questions are stored in an XML log file. SSEA contains a utility program that parses the XML file and converts the data into a tab delimited format for easier statistical analysis. Experiment administrators can modify SSEA to display desired algorithms, colors, graphics and questions.

A full description of SSEA's architecture and capabilities can be found in [42, 63]. Since these publications, SSEA has been updated to administer pre- and post-tests and to play sound. The SSEA log file parser has also been modified to accommodate these changes.

3.5 Conclusion

Before developing these applications, we explored other software used to conduct perceptual and algorithm animation studies. The decision to create our own software packages was reached because we found that existing packages did not provide the exact functionality we desired and that we could control variables by have consistent interfaces for our various types of studies.

Creating the software has allowed for low-level studies to be conducted using VizEval and then, with the same look-and-feel, incorporate these perceptual/attentional features into an algorithm animation in SSEA. These features are then investigated for their influence on the overall comprehension of the presented algorithm. SSEA is additionally used to study interaction and cognitive characteristics of algorithm animations and systems. Now that these software packages have been developed and tested, our experiments can be carried out in a more efficient manner.

CHAPTER 4

PERCEPTUAL AND ATTENTIONAL STUDIES

4.1 Introduction

The algorithm animations studied in this research were developed for pedagogical purposes, meaning they were created for students to gain knowledge about the depicted algorithms. Cognition is the mental process involved in gaining knowledge, including aspects such as awareness, perception, reasoning, and judgment [20]. Therefore, viewers must be able to perceive what is displayed in order to gain knowledge from it. Based on the cognitive load theory [49, 58] and related work [32, 68, 74 85], visualizations that are easier to perceive are capable of freeing some of an individual's limited cognitive processing resources and can allow space within the working memory for processing of other, more complicated tasks.

Attention is defined as the cognitive process whereby a person concentrates on some features of the environment to the relative exclusion of others [46, 89]. "Learning is most efficient when a person is paying attention" [50]. Again, these AAs have been developed for educational environments, so maintaining or guiding a viewer's attention to critical actions in the animation can be vital to his comprehension of the algorithm. Features that are referred to as perceptual and attentional tend to be low-level characteristics of the visualization or animation. For instance, during a sorting algorithm, the use of color to represent objects that are sorted versus those that are not sorted helps the viewer to quickly perceive that objects are grouped into two types. The use of cues or certain motions may be effective in guiding a viewer's attention to specific steps.

Algorithm animations often use rectangular shaped bars or some type of object for which the size indicates the value it represents (i.e. a taller bar represents a larger value). These bars are often changed or moved to depict steps of the algorithm. The objects are sometimes labeled with an alphanumeric value that is intended to add additional clarification to the user. Another form of labels shown on the animation display are those used to provide textual descriptions of the current stage or step of the algorithm. The placement and timing of these labels may have an effect on the comprehension of the algorithm. The studies presented here address some of these low-level features. We are designing and conducting experiments geared specifically towards studying the impact of these individual attributes on algorithm comprehension. The results of these studies will inform the design of effective algorithm animations.

Below is a listing of the perceptual and attentional experiments that have been conducted and analyzed along with the hypothesis, a description of the design and method, and a summary of the results.

4.2 Noticeable Difference Study

The purpose of this initial study was to determine a suitable difference in bar heights that will be noticeable by most viewers. The graphical objects used in the algorithm animations in this research are vertical, rectangular-shaped bars. Since the height of the bar represents its value, we wanted to ensure that viewers would be able to perceive the differences in heights of the bars and hence, the differences in the values the bars represent. By empirically determining this value, other, higher-level features of algorithm animations can be compared, while controlling for the effect of the magnitude of the change in bar height.

The question we sought to answer with this study was: *What is a suitable difference in bar heights that will be noticeable by most viewers?*

4.2.1 Participants

Students enrolled in psychology courses at Georgia Tech participated with this study during the Spring and Fall 2004 semesters. All participants had 20/20 vision after any necessary refractive correction.

4.2.2 Design

This was a discrimination threshold experiment conducted to establish the difference in bar heights to use in the graphical design of future studies. It is classified as a two-alternative, forced-choice judgment because the participants had to select either the left or right bar as being taller, even if the bars appeared to be the same height. The difference, or step size, was determined for the worst-case, viewing bars in the periphery. Since algorithm animation viewers have varying levels of spatial resolution (or ability to detect spatial differences), the step size was selected so that all participants, including those with poorer spatial resolution, will be able to easily perceive the differences between the heights of the bars. The minimum step size was determined by participants perceptually distinguishing one bar height from another with at least 95% accuracy.

4.2.3 Materials

Dell Dimension desktop computers with Sony Trinitron 19" color monitors were used. Participants interacted with the *JND* software package, which managed the graphics and recorded participants' responses. To ensure the individual bars were clearly visible, they were approximately 0.75^o visual angle in width, colored green, and presented against a faint gray background [19]. The bars ranged in height from a set minimum to a set maximum number of pixels, in a constant increment of pixels.

4.2.4 Procedure

Using the *JND* software, participants viewed multiple trials. Each trial consisted of a pair of bars displayed on the top portion of the screen and one question at the bottom asking the participant to judge which bar was taller, the left or the right. The two bars were always a set number of degrees or pixels apart, but their location on the screen varied from the center to the far left or right periphery. Viewers clicked on the button “Left” or “Right” to indicate the bar they judged to be taller, and the responses were stored in a tab-delimited log file.

4.2.5 Results

From a viewing distance of 28.5 inches and a display resolution of 1280 x 1024, analysis of the results found 22 pixels to be a distinguishable difference between two bars even for participants with poorer spatial resolution and for bars in the periphery. This result will aid in building graphical displays that will allow an experimenter to control for a viewer’s ability to perceive height changes and differences.

4.3 Effect of Flash Cue on Detection and Localization

This second experiment was designed to investigate the effect of a flash cue on a viewer’s ability to detect and localize bars that changed height. Many existing algorithm animations use some type of cueing in hopes of drawing the viewer’s attention to an action that is occurring or about to occur. Also, animations often consist of objects resizing or changing shape to indicate some other action or step of the algorithm. These cues and changes take place in the midst of textual descriptions and often in conjunction with other windows with additional views of the algorithm. Investigating whether the cues and changes are perceived as the algorithm animation designer intended would be difficult to analyze within the context of a complete algorithm animation. Therefore, VizEval was used to conduct this study because the display looks similar to the algorithm animations portrayed in SSEA, but only the features of interest (cues and height changes) are shown to the viewer. The questions asked are specific to determining the viewer's ability to perceive the actions taking place.

The question we sought to answer with this study was: *Does the use of a "flash" cue aid in the detection and localization of critical changes?*

4.3.1 Participants

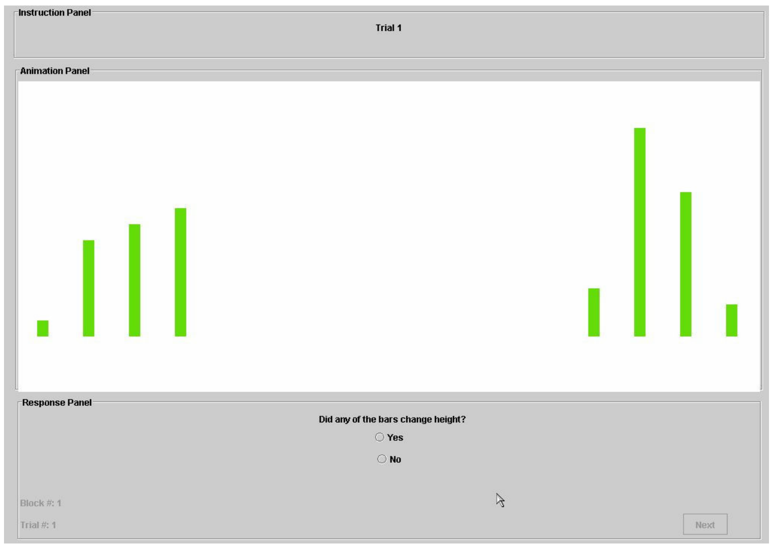
Thirty-six undergraduate students at Georgia Tech participated with this study during the Spring and Fall 2005 semesters. All participants had 20/20 vision after any necessary refractive correction, and all received extra credit that could be allocated towards any psychology course in which they were enrolled during that semester.

4.3.2 Design

This study investigated 4 within-subject variables: labels, set-size, number of bars cued, and number of bars changed. Each participant viewed 576 trials that were divided into 2 blocks of 288 each. One block of trials displayed an alphabetical label placed underneath each bar and, as shown in Figure 4.1, one block did not display labels. The other 3 variables were investigated within each block. The display set-size varied to show 4, 8, or 16 bars. During each trial, either 1 or 2 bars were cued (flashed) followed by 0, 1, or 2 bars increasing or decreasing in height by 22 pixels.

4.3.3 Materials

Participants interacted with the TestTaker module (see Figure 4.1 below) of the VizEval Suite, which managed the graphics and animations and recorded participants' responses. Dell Dimension desktop computers with Sony Trinitron 19" color monitors, 1280 x 1024 resolution, were used.



Did any of the bars change height?

Figure 4.1: The *TestTaker* interface during a session.

To ensure the individual bars were clearly visible, they were approximately 0.75^{o} visual angle in width, colored green, and presented against a faint gray background. The bar heights represent the values of data elements in an array. The bars varied in height from 44 to 374 pixels, in increments of 22 pixels. A preliminary study (described in Section 4.2) showed that an increment of 22 pixels was clearly detectable, even in the far peripheral portion of the screen and for participants with poorer spatial resolution.

4.3.4 Procedure

Students volunteered to participate in this study through an online system. Sessions consisted of one or two students who signed consent forms, read introductory material and had their visual acuity tested before beginning the study. Each participant then used the TestTaker module to complete two blocks of trials, one with labels and one without, and the order of the blocks was balanced across participants. The other three variables (set-size, number of bars cued, and number of bars changed) were randomly varied within each block.

Each trial began with a countdown displayed as a fixation point at the top center of the display window. The countdown began after the user pressed the spacebar and when it ended there was a short pause (a random number of milliseconds) before the start of the animation. Each of the 576 trials consisted of a brief animation followed by five questions. Animations showed either one or two bars flash twice (hide/show for 100 milliseconds, two times), and then zero, one, or two bars were randomly increased or decreased in height by 22 pixels. In trials where two bars were either cued or changed height, the bars were displayed on opposite sides of the screen. This was done to encourage participants to simultaneously monitor both sides of the display.

At the end of each animation, participants answered the following five questions by responding with the mouse:

1. Did any of the bars change height? (Yes or No)
2. Please click on the bar most likely to have changed height.
3. How confident are you that this bar actually changed height? (1 = least …7 = most confident)
4. Please click on the bar second most likely to have changed height.
5. How confident are you that this bar actually changed height? (1 = least … 7 = most confident)

4.3.5 Results and Analysis

The results for detection and localization of changes in bar height differed. Perceptual/attentional characteristics that helped detection hurt localization, and vice versa. First, detection was significantly better when two bars simultaneously changed height than if only one changed ($F(1,33)=62.96$, $p<0.001$); conversely, localization performance was worse when two bars changed height ($F(1,33)=89.18$, $p<0.001$). One possible cause of this result is that two bars changing height doubles the viewer's chances of detecting a change but also doubles the number of locations that must be stored in memory. The location of the second bar is fading from memory while the participant answers questions about the location and confidence level associated with the first bar.

Second, localization performance suffered, as expected, from the set-size effect. Meaning, participants' ability to locate the correct bar that changed height decreased as the number of bars increased from 4 to 8 to 16. However, detection performance was worse for smaller set sizes than larger sets when the bars were located in the periphery and was not affected by set-size effect when the bar locations were random.

The use of labels was a third characteristic that produced differing performance levels for detection and localization activities. While labels significantly improved localization performance ($F(1,33)=7.88$, $p<0.008$), they slightly harmed detection performance. Labels were especially helpful in localizing bars when either two simultaneous changes in bar heights occurred or when 8 or 16 bars were displayed. The labels provided additional cues that helped to retain information about critical bar locations in working memory while they responded to other questions during each trial. The slightly poorer detection performance on trials with labels may have been because the labels provided no critical information for detection, yet doubled the number of objects on the screen and added clutter to the display.

Our results show that some perceptual/attentional characteristics of the animation display may help processing for one type of task, but harm another. Thus, we must systematically consider both the users' perceptual and attentional capabilities as well as the demands of the specific tasks involved in apprehending and comprehending algorithm animations.

4.4 Cueing and Motion in Algorithm Animation Comprehension

Analysis of the flash cue study described in section 4.3 provided insight into the usefulness of a flash cue in detecting and localizing changes on a perceptual and attentional level. This third experiment integrates the flash cue into an algorithm animation and investigates its effect on the comprehension of the depicted algorithm. With this study we were able to investigate if using a flash cue to indicate that two bars were being compared helped the viewer perceive the comparison and, further, if the ability to notice the comparison increased the overall understanding of the algorithm. We also tested a second factor, the type of motion used to illustrate an exchange of values within an array, and again, viewers were tested on their ability to notice this change and its effect on the comprehension of the algorithm.

The question we sought to answer with this study was: *Does the use of exchange motion and comparison cueing aid in the comprehension of algorithm animations?*

4.4.1 Participants

Students were recruited from various undergraduate-level computer science courses at The University of Georgia during the Fall 2005 and Spring 2006 semesters. Fifty-nine volunteers came into our research lab and spent approximately one hour viewing animations and answering questions. Each participant received a five-dollar cash stipend at the completion of the experiment.

4.4.2 Design

Quicksort is a divide-and-conquer algorithm in which array elements are compared to a “pivot” value and then placed into a “lower” (value <= pivot) or “higher” (value > pivot) partition based on the comparison. This study concurrently investigated two factors, cueing and exchange motion, each with two levels. For cueing, the “Flash” group was presented with pairs of bars that flashed three times when they were compared and the bars in the display for the "None" group did not flash when values were compared. For both groups, arrows pointed to the 2 bars being compared. For motion, one group (Move) saw the bars swap positions using an arc-shaped path. The other motion (Grow) showed the bars stay in their positions, but grow or shrink to the size of the other bar.

Table 4.1: 2 x 2 Factorial Experiment Design - Number of participants per group.

	Move	Grow
Flash	14	16
None	12	17

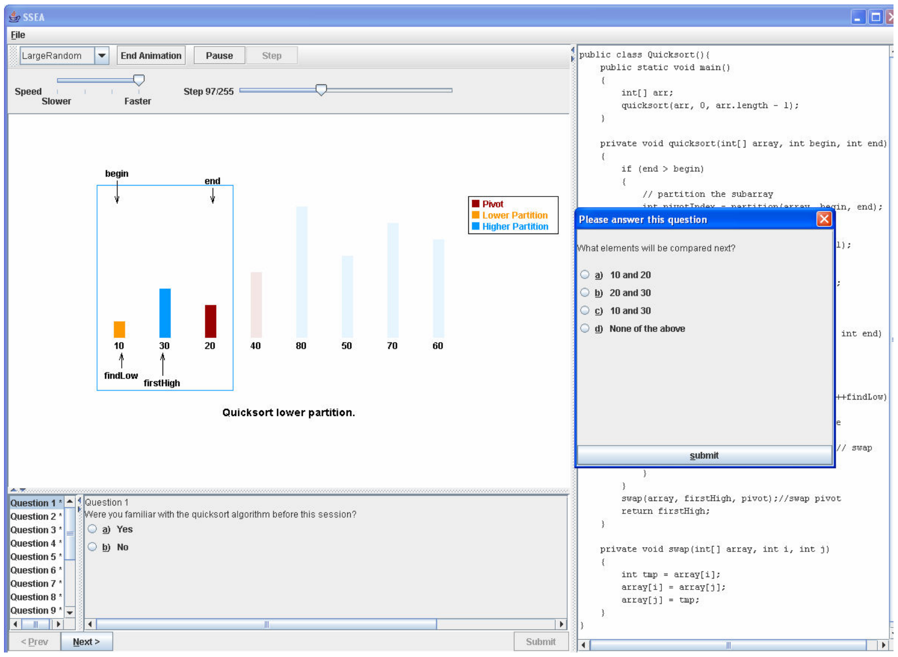
To conduct this study, four animations of the quicksort algorithm were created and participants were randomly assigned to one of the four groups (Move/Flash, Move/None, Grow/Flash, Grow/None).

4.4.3 Materials

Experiments were conducted through the SSEA environment. Each participant was given a packet that consisted of a consent form, an instruction sheet, a SSEA "cheat sheet", scratch paper, and a feedback form. Samples of these materials are in Appendix A. The instruction sheet provided an overview of what was expected of each participant during the experiment and step-by-step instructions of how to use SSEA for the practice exercise. The SSEA "cheat sheet" shows a screen shot of SSEA (see Figure 4.2) along with an explanation of each of the available views. Studies were conducted on Dell Dimension desktop computers with high-resolution 17-inch LCD flat-panel color monitors.

4.4.4 Procedure

Upon arrival to the lab, each participant was given a packet and a brief verbal description of what it contained. Students were randomly assigned to the various groups and were not aware that they were viewing different versions of the animation. They each ran the "SSEA_Demo" of an algorithm that finds the maximum value of the input set. Just as in the actual experiment, the demo contained an animated and pseudocode view of the algorithm and the participants were required to answer one pop-up question and four "traditional" test questions located at the bottom of the screen (see Figure 4.2). The users were instructed to explore the interactive facilities of SSEA with specific, written steps of how to play, pause, or step through the algorithm, answer questions, etc.



Please answer this question

What elements will be compared next?

- a) 10 and 20
- b) 20 and 30
- c) 10 and 30
- d) None of the above

Question 1
Were you familiar with the quicksort algorithm before this session?

- a) Yes
- b) No

```
public class Quicksort(){
    public static void main()
    {
        int[] arr;
        quicksort(arr, 0, arr.length - 1);
    }

    private void quicksort(int[] array, int begin, int end)
    {
        if (end > begin)
        {
            // partition the subarray
            int pivotIndex = partition(array, begin, end);
                                                    1);

                                                    ;

                                                     int end)

                                                    ++findLow)
                                                    e
                                                    // swap
            }
        }
        swap(array, firstHigh, pivot);//swap pivot
        return firstHigh;
    }

    private void swap(int[] array, int i, int j)
    {
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }
}
```

Figure 4.2. Screenshot of the SSEA program with a pop-up question displayed.

After students were comfortable with the system, they ended the demo and began the portion of the study that we used for analysis. The first screen presented was a questionnaire in which students indicated their gender, classification, and every undergraduate computer science course taken in the past or in which they were currently enrolled. Students then proceeded to view an implementation of the quicksort algorithm. We chose to use the quicksort algorithm because of its complex nature, making it a challenging algorithm to understand and follow.

The participants viewed their respective animation and all answered the same fifteen "traditional" test questions at their own pace. All questions were multiple-choice and reflected a range of concepts related to comprehension of the quicksort algorithm. These questions are similar to those on the post-test in Appendix C. The students had access to these questions at all times, and they were allowed to use the animation or code to aid in their responses. Since all interaction data is captured, we may eventually be able to find usage trends that may provide insight for future studies.

Once students completed and submitted the traditional test questions, they were given the opportunity to comment on the animations they viewed, the SSEA system, or give any general feedback, through a paper survey form. After feedback forms were collected, students received payment for their participation.

4.4.5 Results and Analysis

Questions about perception, attention, and comprehension were investigated with this study. To analyze the results, we used the SAS statistical software package to perform one-factor and two-factor ANOVA analyses. Statistical significance at the $\alpha = 0.05$ level was employed.

Perception and Attention

The algorithm animation was periodically paused for users to answer pop-up questions. Each participant was required to answer the same set of eight pop-up questions that were presented immediately following comparison or exchange actions. Four questions were specific to the comparison action:

1 - 2. What elements were just compared? (Asked at two different steps.)

3. What was the last comparison?

4. What variables were compared?

The other four were specific to the swapping action and the two sets of questions were intermixed:

1 - 2. What elements were just swapped? (Asked at two different steps.)

3. What element was just swapped with 70?

4. What variables were swapped?

Table 4.2: Averages for Pop-up Questions

	Move	Grow	AVG
Flash	72.9 %	71.6%	72.2 %
None	62.3 %	58.4 %	60.0 %
AVG	68.0 %	64.8 %	

Table 4.2 shows the average scores, by group, for all eight pop-up questions. Participants in the cueing (Flash) group significantly outperformed the no-cueing (None) group on the pop-up questions overall, F(1,57)=4.44, $p < 0.04$, as well as the comparison specific questions, F(1,57)=10.39, $p< 0.002$. The move group performed slightly better than the grow group overall, and significantly better on the swapping specific questions, F(1,57)=5.74, $p < 0.02$.

Comprehension

Comprehension was measured based on a participant's performance on the "traditional" questions. These questions were designed to cover the first three levels of Bloom's Taxonomy: *Knowledge, Comprehension, and Application* [89]. Analysis of overall performance of the four animation groups on the traditional questions as well as the individual levels yielded results that were not significant.

Table 4.3: Averages for Comprehension Questions

	Move	Grow	AVG
Flash	61.6 %	61.6%	61.6 %
None	59.1 %	60.4 %	59.9 %
AVG	60.4 %	61.0 %	

In conclusion, participants who viewed the flash cue and participants who viewed the arc-shaped motion performed significantly better on the perceptual and attentional questions. However, this benefit did not appear to translate into a higher comprehension of the algorithm. The use of labels and color provided redundant cues to the actions highlighted by flashing and motion, and apparently conveyed adequate information to the viewer. Even though the features investigated in this study did not increase (nor did it decrease) comprehension, participants liked the “blinking” comparisons and animated swaps.

CHAPTER 5

ATTENTIONAL AND COGNITIVE STUDIES

5.1 Introduction

As described earlier, Naps et al. [27] have identified six levels of student engagement that describe various levels of interaction with a visualization tool. In order of increasing engagement, the categories are no viewing, viewing, responding, changing, constructing, and presenting. The types of interaction design addressed by the following studies pertain to the second and third levels of this taxonomy: viewing and responding.

Dynamic questions have been implemented into well-published algorithm animation systems [53, 76, 30] and given as a recommended feature for such pedagogical tools [71, 55]. Several studies were conducted that partially investigate the use of dynamic questions by only considering one aspect, such as prediction [39], or the details were not provided [30].

Interactive questioning techniques can vary based on when and how questions are asked as well as what is asked. Questions can be in the form of a pop-up that stops the animation and forces the user to respond or of what we call "traditional", worksheet-style questions that can be answered at any time with the assistance of the animation or code. Questions may require the user to predict what is about to happen or to tell what they just saw or interpreted. The questions may be low-level, asking about actions of the animation, and affect where the viewer places her attention. Alternatively, the questions may be high-level and require the user to think about the algorithm. We have investigated interactive questions by conducting a series of studies to address these different techniques. We have also begun to explore variations in performance inevitably present in studies involving human subjects by analyzing the effects of individual differences and attention (or visual focus as detected by eye-tracking equipment) on algorithm comprehension.

5.2 Interactive Questioning in Algorithm Animation Comprehension, Part I

Our cueing and motion study described in Section 4.2 tested whether a specific cueing or swapping technique increases a user's ability to notice changes occurring at each step of an algorithm during execution and if greater ability to notice these changes enhances the overall comprehension of the algorithm [42]. During the design phase of the previous experiment, we decided to add pop-up questions as part of the animation of the algorithm. The pop-ups immediately followed one of the comparing or exchanging actions and were intended to help determine whether these techniques increased a student's ability to notice the actions taking place during the animation. After adding the pop-up questions and running pilot tests, we felt that the questions forced us to pay better attention to the animation. Analysis of the results of the cueing and motion experiment found a correlation between performance on the pop-up questions and regular test questions. Therefore, we decided to follow up with this study on interactive questioning techniques.

The question we sought to answer with this study was: *Does the use of interactive questioning aid in the comprehension of algorithm animations? If so, what types and how?*

5.2.1 Participants

Students enrolled in various computer science undergraduate courses were recruited to come into our research lab and spend approximately one hour viewing animations and answering questions. Thirty-four undergraduate students at the University of Georgia participated in this study during the Spring and Fall 2006 semesters. Each participant received a ten-dollar cash stipend at the completion of the experiment.

5.2.2 Design

The purpose of this study was to explore more than just the difference in performance between viewers who interact with the system via pop-up questions and those who do not. We also wanted to determine if providing immediate feedback to the pop-up questions as well as the type of question asked have a significant impact on the performance results. The types of pop-up questions were predictive (What will be the next step?) or responsive (What did you just see?). As shown in Table 5.1, our experiment design consisted of 6 groups. Even though the *no pop-up* group for both *predictive* and *responsive* are the same, we included participants in both of these categories for analysis purposes.

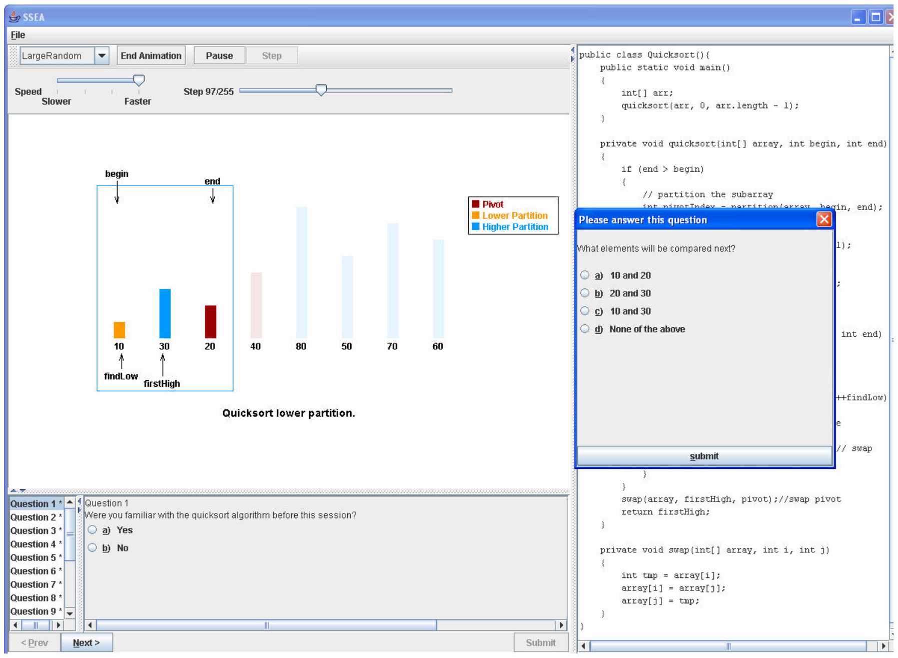
Table 5.1: 2x3 Factorial Experiment Design - Number of Participants Per Group.

	No Pop-up	Without Feedback	With Feedback
Predictive	6	6	5
Responsive	5	6	6

Based on previous studies and intuition, we believed that the participants who were asked to predict the next step of the algorithm through a pop-up question and those who were provided the correct answers to their pop-up questions would perform better on the traditional test questions than participants in the other groups.

5.2.3 Materials

Experiments were conducted through the SSEA environment. Each participant was given a packet that consisted of a consent form, an instruction sheet, a SSEA “cheat sheet”, scratch paper, and a feedback form. A sample of these materials can be found in Appendix A. The instruction sheet provided an overview of what was expected of each participant during the experiment and step-by-step instructions of how to use SSEA for the practice exercise. The SSEA "cheat sheet" shows a screen shot of SSEA (see Figure 5.1) along with an explanation of each of the available views.



Question 1
Were you familiar with the quicksort algorithm before this session?

a) Yes
b) No

What elements will be compared next?

a) 10 and 20
b) 20 and 30
c) 10 and 30
d) None of the above

```
public class Quicksort(){
    public static void main()
    {
        int[] arr;
        quicksort(arr, 0, arr.length - 1);
    }

    private void quicksort(int[] array, int begin, int end)
    {
        if (end > begin)
        {
            // partition the subarray
            int pivotIndex = partition(array, begin, end);
                                                  1);

                                                  ;

                                                   int end)

                                                  ++findLow)
                                                  e
                                                  // swap
            }
        }
        swap(array, firstHigh, pivot);//swap pivot
        return firstHigh;
    }

    private void swap(int[] array, int i, int j)
    {
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }
}
```

Figure 5.1. Screen shot of the SSEA program with a pop-up question displayed.

Studies were conducted on Dell Dimension desktop computers with high-resolution 17-inch LCD flat-panel color monitors.

5.2.4 Procedure

Upon arrival to the lab, each participant was given a packet and a brief verbal description of what it contained. Students were randomly assigned to the various groups and were not aware that they were viewing different versions of the animation. They each ran the "SSEA_Demo" of an algorithm that finds the maximum value of the input set. Just as in the actual experiment, the demo contained an animated and pseudocode view of the algorithm and the participants were required to answer and submit the test questions located at the bottom of the screen (see Figure

5.1). The users were instructed to explore the interactive facilities of SSEA with specific, written steps of how to play, pause, or step through the algorithm, answer questions, etc. The demo did not contain pop-up questions of any form.

After students were comfortable with the system, they ended the demo and began the portion of the study that we used for analysis. The first screen presented was a questionnaire in which students indicated their gender, classification, and all undergraduate computer science courses taken. Students then proceeded to view an implementation of the quicksort algorithm. We chose to use the quicksort algorithm because of its complex nature, making it a challenging algorithm to understand and follow.

The participants viewed the animation and answered fifteen "traditional" test questions at their own pace. These questions were accessible by the students at all times, and they were allowed to use the animation or code to aid in their responses. Since all interaction data (i.e. changes to speed, replays, changing between input sets, selecting a question, etc.) is captured, we hope to eventually be able to identify usage trends that may provide insight for future studies.

The animation for participants in the *no pop-up* group was never automatically paused and users were not required to interact with the AA system. Students in the other groups were required to answer either eight predictive or eight responsive pop-up questions. Half of the participants in each group were provided feedback in the form of the correct choice and a brief description.

Once students completed and submitted the traditional test questions, they were given the opportunity to comment on the animations they viewed, the SSEA system, or give any general feedback, through a paper survey form. After feedback forms were collected, students received payment for their participation.

5.2.5 Results

Several questions were addressed with this study and to analyze the results, we used the SAS statistical software package to perform one and two factor ANOVA analyses. Statistical significance at the $\alpha = 0.05$ level was employed.

Pop-up versus No pop-up

Does the type of interaction required through the use of pop-up questions enhance overall comprehension of an algorithm? Similar to the results by Jarc et al. [39], we found that interactive questioning lessened performance, but not significantly ($F(1,27)=1.49$, $p= 0.233$). The *no pop-up* group had an average score of 76.4% while the *pop-up* groups had a combined average of 67.2%.

The *no pop-up* group scores appear to have been tainted by four participants in that group who completed the experiment during a single session. They spent on average 13.6 minutes viewing the animation and answering the test questions while the average for all participants in all groups (including these four) was 30.7 minutes. Using the Pearson Product-Moment Correlation Coefficient, we found a linear relationship ($r = 0.7266$) between time spent and performance for the *no pop-up* group.

Table 5.2: Average percentage correct per group on the "traditional" test questions.

	No Pop-up	No Feedback	With Feedback	Average (Exludes "No Pop-up")
Predictive	77.8%	70.0%	65.3%	67.9%
Responsive	74.7%	64.4%	68.9%	66.7%
Average	76.4%	67.2%	67.3%	

Table 5.3. Average percentage correct per group on the "pop-up" questions.

	No Feedback	With Feedback	AVG
Predictive	43.8%	65.0%	53.4%
Responsive	77.5%	93.8%	86.4%
AVG	59.1%	80.7%	

Predictive versus Responsive

Does requiring a student to predict the next step of an algorithm during execution enhance overall comprehension of an algorithm? According to the results in Table 5.2, there is not much difference in scores for students who answer pop-up questions that require them to select the next action versus those who respond about the previous step.

The difference in performance on the pop-up questions shown in Table 5.3 are statistically significant (F(1,18)=15.22, *p*=0.0013) but this is expected since predicting what is about to happen requires some level of understanding of the algorithm as opposed to identifying the action that just took place.

Feedback versus No Feedback

Does providing students with immediate feedback in the form of the correct answer and brief description enhance overall comprehension of an algorithm? As shown in Table 5.2, students who received feedback performed better on the regular test questions than those who were only shown the pop-up question, but again the difference was not significant (F(1,18)=0.65, *p*=0.4336). However, providing feedback to the pop-up questions significantly improved scores on the pop-up questions themselves (F(1,18=6.29), p=0.0233).

Use of Color

Our last two experiments were run during the 2005 – 2006 school year and both used the SSEA program to show the same quicksort implementation to participants who were allowed to volunteer for only one of the studies. Due to the high similarity between the materials and procedures of the two experiments, we were able to make comparisons between specific groups from the different studies.

In the animation, color is used to indicate the current status (sorted, active/inactive, lower/upper partition) of a bar, or array element, as the algorithm executes. For this experiment, we reduced the number of colors used in the animation from six to three and used fading to create an appearance similar to that of using additional colors.

Answering the same fifteen test questions, the *responsive-with-feedback* group of this study who viewed the quicksort animation with three colors had an average score of 76% while the comparable group from the cueing and motion experiment who viewed an animation with six colors averaged 61.9%. This difference is nearly significant (F(1,17)=3.59, *p*=0.0754).

User Feedback

Several students commented that the use of colors for grouping partitions was helpful as well as the "blinking comparisons" that we define as cueing. Most of them stated that they really liked the synchronized pseudocode view even though a few followed up by saying that "there was too much to take in at once" or "stimulation overload". Many students also found the speed control, and step and replay functions to be very useful. A few would like more details in the captions explaining the steps while others thought that the captions were distracting.

5.2.6 Analysis and Conclusion

In general, conducting empirical studies of this sort in which students are recruited on a voluntary basis and asked to perform their best on a task that will not count towards a grade will involve outlier scores that contribute to a higher standard deviation and cause the results to not be statistically significant. Even with this factor in mind, the data from this experiment still helps to strengthen previous findings and opens new areas of interest.

As stated earlier, very few experiments have been conducted that focus strictly on the influence of pop-up questions on the comprehension of algorithms. Our results support the findings of [39] in that interactive prediction does not help, and actually lessens, the overall performance on test questions. Jarc et al. attributed the decrease in performance to weaker students treating the interaction as a guessing game, but we believe that the difference may be more related to the user having the opportunity to see the uninterrupted high-level execution of the algorithm without being required to focus on low-level, procedural actions. We investigated this hypothesis in the two experiments discussed in the following sections. First, Section 5.3 describes a study in which we examined performance differences based on question complexity (high- or low-level) and how/when questions were asked and answered (at specific steps via pop-ups or at anytime via traditional questions). Next, we used eye-tracking equipment to capture and explore a user's attention based on visual focal points and describe the results of this data as well as its correlation to performance and individual differences.

If an AA developer chooses to use some type of interactive questioning technique, she may use either predictive or responsive questions since the performance on comprehension questions was nearly the same. However, providing immediate feedback to those questions does appear to be a useful practice.

Using fewer colors requires the user to think less about what each color means and permits the user to focus more on the animation. Hence, a cognitive economy is created that increases learning by minimizing a viewer's mental load while maximizing the amount of information shown [85].

Many users stated that they found the highlighting of the pseudocode in synchronization with the animation to be helpful. We are interested in finding out how much and in what order students focus on the multiple views of the algorithm, another topic investigated through the use of an eye-tracking device and discussed in section 5.4.

5.3 Interactive Questioning in Algorithm Animation Comprehension, Part II

For the first interactive questioning experiment, we expected the group of students with the dynamic questions to perform better on the comprehension questions, but since they performed worse, we investigated this phenomenon further. One possible explanation for the poorer performance of the group with pop-up questions is that the low-level questions that were asked may have caused the participants to focus on the actions of the animation and not the overall execution of the algorithm. This possibility is explored in this study by asking participants questions with different levels of complexity.

This second interactive questioning study separates traditional questions from pop-up questions. Traditional questions can be considered interactive because the viewer is able to use the animation and code to answer the questions. All of the pop-up questions used in this study were responsive with feedback since the first interactive study (see Section 5.2) found no difference in comprehension performance for the various groups. For each set of interactive questions, one group of participants was asked low-, procedural-level questions and the other high-, comprehension-level questions. The control group did not receive any questions during the execution of the algorithm. In addition to viewing the animation, all participants completed a pre-test, a questionnaire, a post-test, and a series of individual preferences and abilities tests.

The questions we sought to answer with this study were: *Does the use of interactive questioning aid in the comprehension of algorithm animations? If so, what types and how?*

5.3.1 Participants

Forty-four students enrolled in CS 1331, *Introduction to Object-Oriented Programming,* offered at Georgia Tech during the Fall 2006 semester participated in this study. Each participant received extra credit in that course and a five-dollar stipend. As an alternative, students who wished to earn extra credit without participating in the study could prepare a presentation containing an overview of the Mergesort algorithm, step-by-step implementation details, an analysis of runtime, and a comparison to other algorithms.

5.3.2 Design

As shown in Table 5.4, our experimental design consisted of two factors: level of question complexity and the method by which the questions were asked during the animation. We divided questions into two levels of complexity, low or high. The low-level questions simply required the user to read or recall information presented on the screen. Correctly responding to the high-level questions required understanding of the algorithm or functions executing within the algorithm.

Table 5.4: 2x3 Factorial Experiment Design - Number of participants per group.

	No Questions	Traditional	Pop-up
Low	8	7	8
High	7	7	7

Even though the *No Questions* group for both *Low* and *High* are the same, we included participants in both of these categories for analysis purposes.

Some participants were not required to answer questions during the animation ("No Questions"). Those who were required to interact with the AA were presented with questions either in the form of pop-ups or an onscreen worksheet ("Tradional"). The traditional questions asked for details about the algorithm and were always available at the bottom of the screen. The pop-up questions paused the animation at a particular step and required the user to respond before proceeding with the animation.

For this study we collected data pertaining to individual differences between the learners and have begun to investigate whether algorithm animations may be better suited for students with certain learning preferences or with particular inductive, spatial, or memory capabilities.

5.3.3 Materials

The materials used for this study were the same as those for the first interactive questioning study described in Section 5.2 plus a pre-test (see Appendix B) and a battery of individual assessment tests. The names, descriptions, and purposes for each of these tests are as follows:

Surface Development (VZ-3) [23]

The Surface Development assessment tests a participant's spatial ability by requiring him to envision manipulating a three-dimensional object and to answer questions about the finished object. The test used in this study specifically asked the participants to imagine folding a piece of paper into an object/cube and then match the edges of the flat piece of paper with the edges of the object.

Figure Classification (I-3) [23]

The Figure Classification assessment tests an individual's inductive reasoning capabilities. Participants were given a handout that contained several sets of figures in which they had to recognize similarities and differences. Each problem contained 2 or 3 groups of figures with 3 items each. The participant had to determine features that were similar amongst figures within each group and different from the other groups and then classify ungrouped figures based on their findings.

Backward Digit Span (Working Memory)

The Backward Digit Span assessment was used to measure each participant's working memory. The facilitator reads a series of 2 to 8 numbers aloud to the participant who then writes the numbers in backwards order after the facilitator finishes saying all 2 to 8 numbers of that series. For example, the facilitator says 5 2 8, and the participant is expected to write 8 2 5, then the facilitator says 9 3 1 5 7 4 3 and the participant attempts to write 3 4 7 5 1 3 9.

Learning Styles Inventory

A learning styles inventory described in [25] and available at http://www.engr.ncsu.edu/learningstyles/ilsweb.html categorizes students into groups along 4 different dimensions: Visual-Verbal, Reflective-Active, Intuitive-Sensory, and Global-Sequential. Participants answer 44 two-choice questions, 11 from each of the 4 groups, and are classified accordingly. For instance, if a participant selects all visual responses for the 11 questions in the Visual-Verbal category, then he will be self-classified as strongly visual. However, if another participant answers 5 verbal and 6 visual, then the difference of 1 would result in him being classified as a visual learner, but not having a high preference for either.

Visual Acuity

The computerized Vision Acuity test is a quick way to determine if a viewer is visually impaired. We want to eliminate this as a factor that could influence performance on the various test, especially the algorithm animation comprehension test. To complete this test, each participant sits a specified distance from the computer monitor and reads aloud the characters shown on a chart on the screen.

Color Perception Deficiency Screening

Deficiencies in color perception were screened using Ishihara Plates.

5.3.4 Procedure

The procedure for this experiment was basically the same as the first interactive questioning study. This one also included written tests for the surface development, figure classification, and backward digit span assessments. The overall flow for each participant was to sign a copy of the consent form, take a timed, written surface development and figure classification tests.

Students were then able to work at their own pace on the computerized portion of the study in which they completed the SSEA demo, questionnaire, pre-test, algorithm animation, post-test, and learning styles assessment. As each one finished, the backward digit span, visual acuity, and color perception deficiency tests were administered on a one-by-one basis.

5.3.5 Results and Analysis

Algorithm comprehension was measured by performance on pre- and post-tests; a subset of the questions in the post-test were asked as the pre-test. The difference between each participant's score on that subset of post-test questions and the pre-test was recorded as performance improvement.

Performance by Question Type and Level

Tables 5.5 – 5.8 show test and improvement scores by group. The values displayed in the fifth column of each table are the average scores for the groups who were asked low- or high-level, pop-up or traditional questions.

Even though the students were randomly assigned into one of the six groups, the pre-test scores indicate a nearly significant difference, (F(1,38)=3.4902, p=0.06946), in previous knowledge about the quicksort algorithm between the low-level (average score of 44.17%) and high-level (60.71%) groups. Also, the no questions group (59.17%) scored moderately higher than the pop-up group (49.17%). The group types had no effect on the pre-test scores since the users had not viewed any versions of the animation. This improbable circumstance of random assignment yielding groups with nearly significant differences in prior knowledge impedes the analysis of performance based on group type. However, some inferences can be drawn, and fortunately, other data was gathered about participants' individual abilities.

Though large differences exist between the pre-test scores of some groups, most of these differences have been eliminated by the post-test (see Table 5.6). This trend is especially noticeable within the "Traditional Only" group. Pre-test scores differed by 17.86% whereas the post-test scores on those same questions differed by only 1.79%, a 32.14% improvement for the students who answered low-level questions versus only a 16.07% increase for those who answered high-level traditional questions. A one-tailed t-test produced a p-value of 0.164638.

Another observation deals with the "Pop-up Only" group. Pre-test scores show that the students in the "High" group (42.19%) had more prior knowledge of the algorithm than those in the "Low" group (57.14%) with almost a +15% advantage and less room for improvement. However, their scores improved by 28.57% while the "Low" group only improved by 23.44%, meaning that high-level pop-up questions appear to be advantageous in increasing algorithm comprehension, though not significant.

Table 5.5: Pre-test scores. Random assignment generated "High" versus "Low" groups with nearly significant differences in prior knowledge of the algorithm. (F(1,38)=3.4902, p=0.06946).

	No Questions	Pop-up Only	Traditional Only	AVG (Excludes "No Questions")
Low	55.36%	42.19%	46.43%	44.17%
High	62.50%	57.14%	64.29%	60.71%
AVG	59.17%	49.17%	55.36%	

Table 5.6: Post-test (subset) scores. These are average scores on the subset of post-test questions that were presented as the pre-test. Notice the "Low-Traditional Only" scores are now only 1.79% less than the "High-Traditional Only"

	No Questions	Pop-up Only	Traditional Only	AVG (Excludes "No Questions")
Low	76.79%	65.63%	78.57%	71.67%
High	79.69%	85.71%	80.36%	83.04%
AVG	78.33%	75.00%	79.46%	

Table 5.7: Improvement as a difference between pre-test and a subset of post-test scores. A one-tailed t-test of the difference between the "Traditional Only" "Low" versus "High" groups produced a p-value of 0.164638.

	No Questions	Pop-up Only	Traditional Only	AVG (Excludes "No Questions")
Low	21.43%	23.44 %	32.14 %	27.50 %
High	17.19%	28.57 %	16.07 %	22.32 %
AVG	19.17%	25.83 %	24.11 %	

Table 5.8: Post-test scores for <u>all</u> questions. No statistically significant differences.

	No Questions	Pop-up Only	Traditional Only	AVG (Excludes "No Questions")
Low	70.54%	67.97%	70.54%	69.17%
High	71.09%	80.36%	74.11%	77.23%
AVG	70.83%	73.75%	72.32%	

The Pearson Product-Moment Correlation Coefficient, *r*, is a measure of the tendency of a pair of variables to increase or decrease together [89]. The value range of *r* is between -1.0 and +1.0 with |*r*| values close to 1.0 considered a strong correlation and values close to 0.0 to be weak. Table 5.11 shows *r* between each pair of tests. A fairly strong positive correlation was found between traditional and post-test performances (r = 0.746192) and a moderate linear relationship between scores on the pop-up questions and post-tests (r = 0.440713). So, students who performed better on the interactive questions tended to also perform better on the post-test.

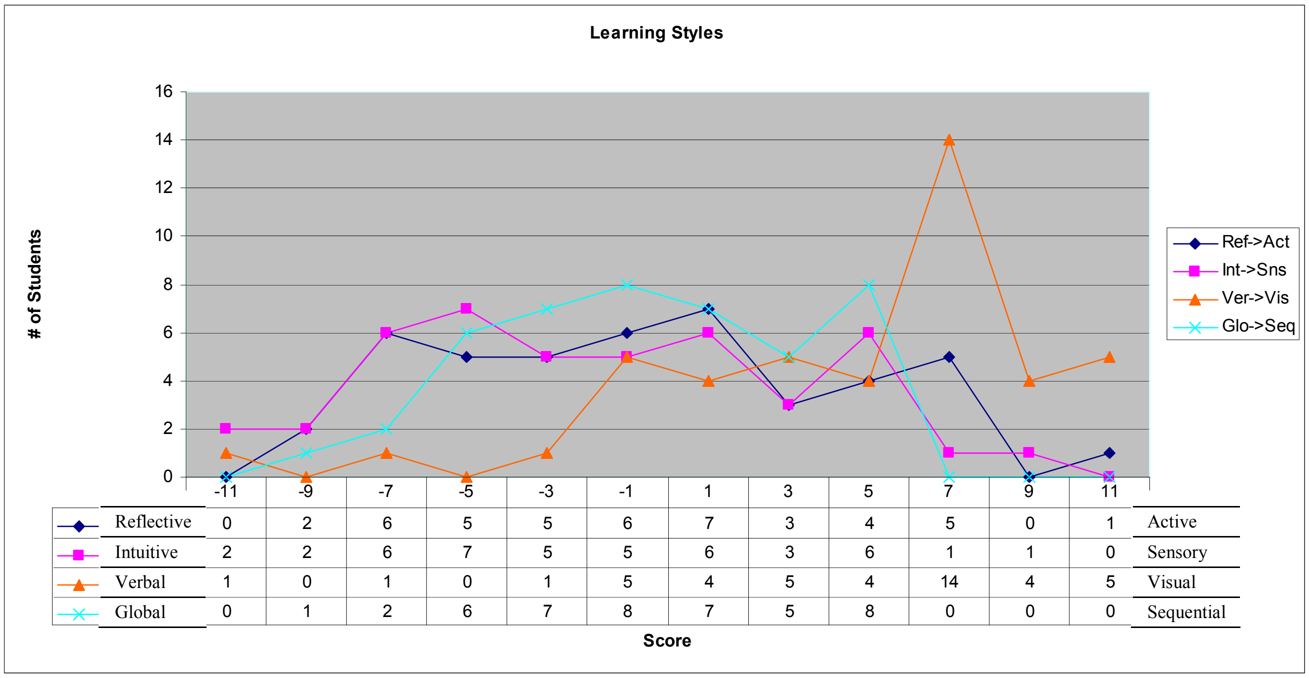
Learning Styles

As described earlier, the learning styles assessment consists of 44 two-choice questions, 11 for each of the 4 categories (Visual-Verbal, Reflective-Active, Intuitive-Sensory, and Global-Sequential). To understand how the values were computed, imagine the scores for each category as a number line [-11, 11]. Starting at 0, each response moves the participant one score to the left or right. If a participant responds in one "direction" for all 11 questions for one category, then his score will be either +/- 11. If he answers 5 in one direction and 6 in the other, his score will be +/- 1, and he will be classified accordingly. There is no significance to which level of a category is + or -; this signage was used only for grading, plotting, and comparison purposes. The chart in Figure 5.2 displays the distribution of scores on the learning styles assessment in which negative scores are for Reflective, Intuitive, Verbal, and Global learners and the positive scores are for Active, Sensory, Visual, and Sequential.

Table 5.9 shows the number of participants, improvement, and post-test scores per learning styles category. Students are almost evenly balanced (20 to 24) in the active/reflective and sequential/global groups. Post-test scores are essentially the same for all groups when viewed as a whole, but further dividing the groups into levels based on their scores [±1, ±11] within each category yields differences. Specifically, the 9 students in the lower end of the sensory group (scores of +1 or +3) had a significant improvement in comparison to the 10 in the lower end of the intuitive group (scores of -1 or -3), P(T<=t) =0.036583. This implies that AAs may be better suited for intuitive learners who are good at grasping new concepts and prefer to learn by discovering [25].

Table 5.9: Improvement and post-test scores by Learning Style. Indentation and shading are used to group the pairs of learning styles per category.

LS - Group	Count	Improve-ment	Post-All
Reflective	24	25.52%	73.96%
Active	20	20.00%	70.31%
Intuitive	27	20.37%	72.69%
Sensory	17	27.21%	71.69%
Verbal	8	15.63%	73.44%
Visual	36	24.65%	72.05%
Global	24	21.88%	72.40%
Sequential	20	24.38%	72.19%



	-11	-9	-7	-5	-3	-1	1	3	5	7	9	11	
Reflective	0	2	6	5	5	6	7	3	4	5	0	1	Active
Intuitive	2	2	6	7	5	5	6	3	6	1	1	0	Sensory
Verbal	1	0	1	0	1	5	4	5	4	14	4	5	Visual
Global	0	1	2	6	7	8	7	5	8	0	0	0	Sequential

Figure 5.2: Distribution of Learning Styles Scores.

A majority of participants were classified as being more visual learners (36) than verbal (8), and achieved a considerably higher level of improvement between pre- and post-test scores, though not significant (P(T<=t) = 0.180749). On average, the visual learners improved by 24.65% while the verbal learners only improved by 15.63%

Differences in performance per learning style were much more evident for the interactive questions themselves. Table 5.10 summarizes the count, the score, and select p-values for the pop-up questions and the traditional questions by complexity level. P-values are shown only for pairs of *low* or *high* scores for a particular learning style's category in which scores differed by a substantial amount and multiple students were classified. For instance, 8 students were in the *popup-low* group, and for the intuitive/sensory category, 6 students were classified as intuitive while 2 were sensory. The average scores shown in the Score Low column were 85.42% and 100% respectively, and the difference was nearly significant with a *p*-value of 0.08124 shown in the P-Value Low column.

The intuitive learners outperformed the sensory learners on the low-level traditional questions with a score of 96.88% to 62.50% and a *p*-value of 0.0134. A similar trend was found for the high-level pop-up questions with scores of 87.50% to 70.83% and a *p*-value of 0.0955. However, the sensory group, who pay attention to detail and are "good at memorizing facts" [25], performed better on the low-level pop-up questions that asked for information about what was just shown. The step-by-step nature of algorithms and hence the animation may contribute to the higher performance on the traditional questions for the sequential learners (85.00%) over the global group (70.83%), $p = 0.1714$.

The observed differences in traditional test scores shown in Table 5.10 account for the low to moderate correlations between performance on traditional questions and each of the learning styles. These correlation values range from 0.2062 to 0.3472, irrespective of positive or negative. Another moderate correlation was found between learning styles, Sensory-Intuitive and Sequential-Global (r = 0.4052). This means that students who are sequential learners have a slight tendency to also be sensory learners, and the same for intuitive and global.

Table 5.10: Interactive question scores by Learning Style. A summary of performance differences between learning styles within a *low* or *high* level interactive questioning group.

LS – Group	Count Low	Count High	Score Low	Score High	Avg	P-Value Low	P-Value High	P-Value Avg
POP-UP SCORES								
Reflective	3	5	91.67%	80.00%	84.38%			
Active	5	2	87.50%	81.25%	85.71%			
Intuitive	6	4	*85.42%*	*87.50%*	86.25%	0.08124477	0.09548615	
Sensory	2	3	*100.00%*	*70.83%*	82.50%			
Verbal	N/A	1	N/A	62.50%	62.50%			
Visual	8	6	89.06%	83.33%	86.61%			
Global	5	3	*92.50%*	79.17%	87.50%	0.17452585		
Sequential	3	4	*83.33%*	81.25%	82.14%			
TRADITIONAL - SCORES								
Reflective	6	2	79.17%	*75.00%*	78.13%			
Active	1	5	100.00%	*67.50%*	72.92%			
Intuitive	4	5	***96.88%***	*67.50%*	*80.56%*	**0.01344336**	0.39278227	0.19174695
Sensory	3	2	***62.50%***	*75.00%*	*67.50%*			
Verbal	1	2	100.00%	*93.75%*	*95.83%*		0.08934513	0.0671377
Visual	6	5	79.17%	*60.00%*	*70.45%*			
Global	5	4	80.00%	*59.38%*	*70.83%*		0.15815798	0.17143403
Sequential	2	3	87.50%	*83.33%*	*85.00%*			

Other Individual Differences

The other tests of individual differences were surface development, figure classification, and backward digit span used to classify participants' capabilities in spatial visualization, inductive reasoning, and working memory. Some noticeable observations from these tests were a few moderate correlations with other performance metrics. For instance, participants with higher spatial capabilities were likely to be more global learners and also had higher inductive reasoning scores (r = 0.448) and post-test scores (r = 0.342). Additionally, students with higher inductive reasoning skills tended to have higher scores on the traditional test questions.

User Feedback

Students again expressed that they liked the step-by-step code highlighting and the speed control and rewind functionalities. When asked for specific opinions about the use of interactive questions, most students preferred the questions because they made them more comfortable, "focus" on the animation, and clarified what was happening. One found the questions to be "annoying" and another found them helpful, but felt they "tended to interrupt *(his)* learning". A few students recommended adding sound in the form of a verbal description so that their visual focus could be more on the animation and not split between the animation and captions. SSEA has already been extended to play sound, and an initial study is being prepared.

Table 5.11: Pearson Product-Moment Correlation, *r*, for All Tests. The value range of *r* is between -1.0 and +1.0 with |*r*| values close to 1.0 (red and bold) considered a strong correlation, values close to 0.0 (faint green) to be weak, and values in midrange (yellow and bold) to be moderate.

	Surf. Dev	Fig. Class.	Bwrd Digit	Pre-Test	Impr-Diff	Impr-Ratio	Post-Test (ALL)	Post (Pre-subset)	Pop-Up	Trad	LS Act/Ref	LS SNS/Int	LS Vis/Vrb	LS Seq/Glo
Surf. Dev	**1.000**	**0.448**	-0.043	0.261	-0.044	0.064	**0.342**	0.242	-0.062	-0.176	0.041	-0.161	0.109	**-0.301**
Fig. Class.		**1.000**	0.228	-0.075	0.187	-0.232	0.241	0.141	0.268	**0.328**	0.112	-0.146	0.181	-0.048
Bwrd Digit			**1.000**	-0.091	-0.046	0.063	-0.036	-0.160	0.080	-0.197	-0.037	-0.005	-0.091	0.173
Pre-Test				**1.000**	**-0.640**	**0.815**	0.248	**0.365**	-0.065	-0.132	0.036	-0.050	-0.263	-0.239
Impr-Diff					**1.000**	**-0.779**	**0.443**	**0.482**	0.123	**0.459**	-0.043	-0.039	0.151	0.054
Impr-Ratio						**1.000**	-0.085	-0.015	0.027	**-0.445**	-0.070	0.085	-0.206	-0.191
Post-Test (ALL)							**1.000**	**0.819**	**0.441**	**0.746**	-0.176	-0.117	-0.092	-0.128
Post(Pre-subset)								**1.000**	0.062	**0.582**	-0.011	-0.104	-0.117	-0.207
Pop-Up									**1.000**	**N/A**	0.030	-0.046	0.200	-0.237
Traditional										**1.000**	-0.239	**-0.347**	-0.294	0.206
LS Act/Ref											**1.000**	-0.156	0.236	-0.244
LS SNS/Int												**1.000**	0.078	**0.405**
LS Vis/Vrb													**1.000**	-0.299
LS Seq/Glo														**1.000**

5.4 Eye-Tracking Study I: Correlation Between Attention, Comprehension, and Individual Differences

Results from our first interactive questioning study showed that students who actively engaged with the animation via responding to pop-up questions had lower test scores. One possible reason for this outcome could be that the pop-up questions encouraged students to focus on specific actions of the animation and not comprehend the overall implementation of the depicted algorithm.

One goal of this first eye-tracking study is to determine if a relationship exists between the type of interactive questions asked and where the viewer places visual focus. Specifically, if the viewer is required to answer low-level questions, will she shift her attention mostly to low-level features such as object labels? If interactive questions are asked, will the viewer focus on regions of the animation in a different way than if no questions are asked? As in the second interactive questioning study, we administered assessments to differentiate students based on preferred learning styles and abilities such as inductive reasoning, working memory, spatial visualization, and fluid intelligence.

The questions we sought to answer with this study were: *Does the type of interaction or task required of a user affect which views and features are attended to and the time spent attending those views and features? Is there a correlation between comprehension and gaze patterns? Between gaze patterns and individual preferences and differences?*

This section provides an overview and summary of our first eye-tracking study. A complete description of the participants, design, procedure and materials can be found in [40].

5.4.2 Overview

Data from nine undergraduate, computer science students at The University of Georgia was analyzed for this study. All participants were enrolled in the CSCI 4800/6800 Human-

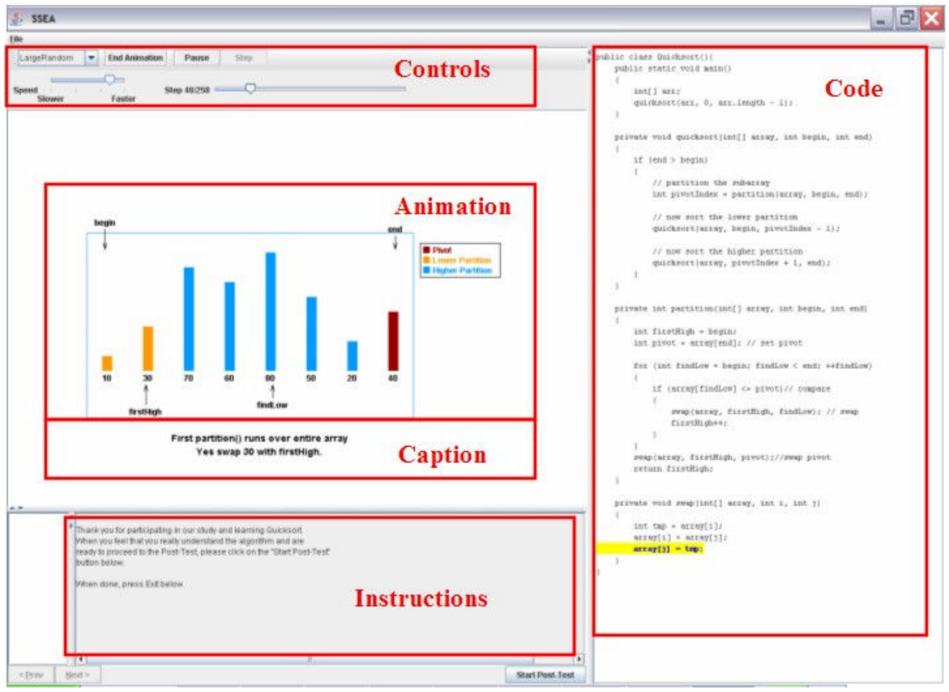
Computer Interaction course during the Spring 2007 semester and received credit in that course for participation in the study. Students were randomly assigned into two groups. Using SSEA, one group of five students interactively viewed an implementation of the quicksort algorithm with pop-up questions, and the other group of four viewed a version of the algorithm without pop-up questions. The data collected included written and computer-based tests designed to assess algorithm comprehension and individual differences. Each participant's eye-movement was captured using the ASL Eye-Trac6000 eye tracking system.

5.4.3 Results and Analysis

Pop-up versus No-pop-up

Algorithm comprehension was measured by performance on pre- and post-tests. A subset of the questions in the post-test was presented to the students as the pre-test. The difference between each participant's score on that subset of post-test questions and the pre-test was recorded as performance improvement. Reinforcing the trend detected in the first interactive questioning study, performance improvement was greater (15.63% versus 5.00%) for students in the no-pop-up group, who viewed an uninterrupted animation of the quicksort algorithm.

As shown in Figure 5.2, the SSEA screen was divided into five areas of interest (AOI): control, animation, caption, post-test, and code. We analyzed the periods of time spent viewing each AOI, or *fixation duration*, and the order in which they were viewed, or *gaze pattern*. During the learning phase, all participants spent the majority of their time watching the animation (average of 48.68% of fixation time), followed by the caption area (24.00%), then the code (14.33%), and the other 13.00% of time was split between viewing the controls or the question areas. The differences between the pop-up and no-pop-up groups in time spent viewing the various AOIs was not statistically significant.



First partition() runs over entire array
Yes swap 30 with firstHigh.

Thank you for participating in our study and learning Quicksort.
When you feel that you really understand the algorithm and are ready to proceed to the Post-Test, please click on the "Start Post-Test" button below.

When done, press Exit below.

```
public class Quicksort(){
    public static void main()
    {
        int[] arr;
        quicksort(arr, 0, arr.length - 1);
    }

    private void quicksort(int[] array, int begin, int end)
    {
        if (end > begin)
        {
            // partition the subarray
            int pivotIndex = partition(array, begin, end);

            // now sort the lower partition
            quicksort(array, begin, pivotIndex - 1);

            // now sort the higher partition
            quicksort(array, pivotIndex + 1, end);
        }
    }

    private int partition(int[] array, int begin, int end)
    {
        int firstHigh = begin;
        int pivot = array[end]; // set pivot

        for (int findLow = begin; findLow < end; ++findLow)
        {
            if (array[findLow] <= pivot)// compare
            {
                swap(array, firstHigh, findLow); // swap
                firstHigh++;
            }
        }
        swap(array, firstHigh, pivot);//swap pivot
        return firstHigh;
    }

    private void swap(int[] array, int i, int j)
    {
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }
}
```

Figure 5.2: SSEA – Eye-tracker Areas of Interest

Another finding was that students frequently switched attention back and forth between the animation and the caption areas with switches between the animation and code being the next most frequent pattern. This means that students are taking advantage of the additional, verbal descriptions when the animation alone may not be sufficient. Again, the differences in switching patterns between the two groups was not significant and implies that the use of pop-up questions does not overall influence which views and features are attended to and the time spent attending those views and features. However, a closer examination of the data does show increases in attention to certain AOIs immediately following pop-up questions in which the answers can be found in those AOIs where attention was shifted. This means that pop-up questions can be used to temporarily guide a viewer's attention to specific actions or AOIs but that they may not have an effect on the overall viewing behavior.

Individual Differences and Viewing Patterns

Several strong correlations were discovered between learning styles and fixation durations. We found that students who self-identified themselves as reflective learners spent more time viewing captions (correlation score of 0.82) while more active learners spent less time (-0.94). Moderate correlations between reflective learners and time spent viewing code (-0.6) and between active learners and time spent viewing code (0.55) and the animation (0.47) indicate that reflective learners, who prefer learning by thinking, used captions to comprehend the algorithm while the active learners, who prefer to learn by doing, viewing the animated graphics and code. Students who were categorized as sequential (step-by-step) learners preferred to view the source code (0.801) while the global (big view) learners spent more time viewing the animation (0.858).

Viewing Behavior Model Graph (VBMG) Clustering

Viewing Behavior Model Graph (VBMG) Markov model based clustering [1] was used to group participants based on their viewing behaviors. VBMG produced three clusters in which an apparent difference in viewing patterns for code and captions was observed while not much difference was found in the viewing patterns of the animation. Results of this clustering showed that viewers who preferred to view code over caption had stronger spatial visualization abilities and those who preferred to view caption over code performed better on the reading span and inference tests.

The results of this initial eye-tracking study have begun to provide insight into the role that individual preferences and abilities may impact performance and algorithm animation design. Developers of AAs should not only consider the type of technology available when creating a visualization, but should also consider the desired goal of the system, and variations in the users' learning styles and other abilities.

CHAPTER 6

SOFTWARE VISUALIZATION ONTOLOGY

6.1 Introduction

The second major area of this research addresses evaluating the effectiveness of a software visualization system which is currently a subjective process. Opinions of the usefulness of a system may be influenced by factors such as who or where it was developed or the number and types of displays available. A system may be deemed effective because students who used it performed better on some type of test than a group of students who were given an alternative, supposedly comparable method of learning the same material.

The ultimate goal of this portion of our research is to provide an objective rating method that can be used to score a software visualization or system based on specific, empirically supported criteria. The higher the score, the more effective, and thus, the faster and more in-depth a user will understand the intended concept. Adequately determining the pedagogical significance of every feature would require years of research by an individual or a group. Proper investigation will require that each of hundreds of features, as well as combinations of features, be isolated in a customized software visualization and then studied with human subjects. The results of these experiments can then be used to give each feature a value that will in turn contribute to the score for the visualization or system of which it is included.

This research has started to scratch the surface of this enormous task by using the results of our studies to categorize a feature as helpful or not. As more studies are conducted, the extent to which each of the features contribute to the overall effectiveness of a system will become more clear and hence provide a numerical value that is a weight of the level of usefulness of these features.

6.2 Background

An objective method for evaluating algorithm animations and the systems that support their creation and execution should involve de-featuring, classifying, and then scoring based on empirically determined weights of the usefulness of specific features. These weights can be used to rate an algorithm animation based on whether it properly implements the assessed features and to rate an algorithm animation system based on whether it is capable of producing the features. This section includes a summary of projects geared towards evaluating or storing information about algorithm visualizations or systems followed by descriptions of a visualization ontology schema and related software visualization taxonomies.

Visualizations and visualization software resources are plentiful and are often found as a list on an individual's or research group's website. Borner and Zhou [6] present a survey of available Information Visualization repositories and resources that include information about some SV systems. Several organizations have developed websites that contain information about SVs as a minor part of an extensive educational resource for computer science and mathematics such as the Computer Science Teaching Center [16], the Math Archives[83], and Netlib [88].

The need for and difficulty of composing an objective evaluation process for algorithm animations has been acknowledged [9, 60] and plans have been made and partially implemented to evaluate, classify, and/or store information about algorithm animations and algorithm animation systems [12, 55]. The Complete Collection of Algorithm Animations (CCAA) [12] and Algorithm Animation Repository (AAR) [55] are two examples such algorithm animation repository projects.

CCAA provides an extensive listing of algorithm animations that can be run over the internet as well as brief descriptions and links to each. **Note: This website, http://www.cs.hope.edu/~alganim/ccaa/, was available through 2006, but has been moved to a different server and will be re-instated (January 22, 2007 - personal communication).* The requirement that AAs listed on this site must be capable of running over the internet does not allow for the inclusion of stand-alone AA systems. The website grouped AAs by the type of algorithm executed, and users were required to browse through the lists in order to find desired animations.

Eight attendees of the 2001 Dagstuhl Seminar on Software Visualization (Seminar No. 01211, May 20-25, 2001) formed a group to build an Algorithm Animation Repository that would serve as a means for algorithm animation developers to "publish" their work [17]. They explain that developers, especially of small-scale systems, are not rewarded for their efforts due to the lack of availability to the general public if the system is not documented in a published paper. In hopes of raising the level of acceptance of algorithm animation software, they planned to have a Board of Editors who would referee and rank entries made into the repository. However, the URL [54] given as the location of the AAR indicates that their work has since shifted to improving the educational impact of algorithm visualizations.

Participants in a workshop held at the UK National e-Science Centre in April 2004 discussed the need for an ontology for visualization [8, 21]. Examination of their proposed categories (*task and use, representation, process, and data*) indicates that we could easily incorporate their high-level categories into our schema and position our SV classes as subclasses to their categories. In 2005, they followed up with an article [22] indicating the motivation for "seeking a more rigorous foundation for visualization" includes collaboration, composition, preservation (curation), and education. They explain that meanings/language within a community can be expressed in different ways: *terminology* (definitions); *taxonomy* (definitions plus structure); and *ontology* (meanings, plus relationships that are machine processable).

The software visualization portion of our visualization ontology schema emerged from the reconciliation of six of the most referenced taxonomies in the SV domain. Below is a summary of each of these six, in chronological order.

Myers [52] discusses the six combinations created by pairing the components of 2 axes – what is shown (code, data, or algorithm) and how it's shown (static or dynamic). For example, a static display of an algorithm may consist of snapshots depicting different stages of execution while a dynamic display would show the algorithm completing in a smooth, continuous motion.

Brown [9] examines the nature of algorithm animation displays. He uses the terms *content* as a category to describe the level of complexity of what is displayed, *persistence* to explain how much history of the data is shown, and *transformation* to represent how changes to data are shown. He also discusses whether views can be used in different situations and termed this category *reuse of views.*

Roman and Cox [69] developed a taxonomy that consists of *scope*, *abstraction*, *specification method*, *interface*, and *presentation*. The aspect of the program being visualized is the domain or *scope* of the system. Similar to Brown's content, *abstraction* describes the level of complexity involved in creating a display, which can consist of simple, direct representations through more abstract, synthesized representations and is set by a user via the specification method. The viewer is impacted by the last two categories in that the *interface* is how they will interact with the system and the *presentation* is what they will see.

Price et al. [60] define six major categories: *scope*, *content*, *form*, *method*, *interaction*, and *effectiveness*. Their use of *scope* is different from that of Roman and Cox [69] in that it refers to the range of programs that the SV system can take as input rather than the aspect of the program being displayed, which Price et al. describe as *content*. The general appearance and how a user instructs the system are classified by *form* and *interaction*, respectively. *Method* is how the animator must specify the desired display, and the *effectiveness* of a system is rated based on how well it communicates information to the viewer. They qualitatively rank the *effectiveness* of a system based on its purpose, appropriateness and clarity, empirical evaluation, and production use.

The tasks encountered during the development of a visualization of a parallel system were addressed by Stasko and Kraemer [41]. The focus is on parallel debuggers, performance evaluation systems, and program visualization systems. Development is broken into three stages: *data collection*, *data analysis*, and *display*. First, *data collection* is accomplished by a form of software or hardware instrumentation. Then the *data analysis* stage involves calculations, ordering of events, and inferences about higher-level events based on the low-level data collected, which may be kept in short-term or long-term storage. *Displays* can use graphs, nodes, coloring schemes, and layouts to show communication between processors, the order of execution of events, or other features of a program.

Stasko and Patterson [80] focus on graphical views of computer programs. Their taxonomy describes the level of each of four dimensions: *aspect*, *abstractness*, *animation*, and *automation*. As [52, 60, 69] have also categorized, *aspect* is the part of the program that is being displayed. *Abstractness*, similar to [69]'s abstraction, describes the level of customization of the visualization to the data being shown, and *automation* deals with the amount of input expected by the animator to create the visualization.

While researching how others have classified SV systems, we found many discrepancies in definitions and use of terminology. For instance, categories in various taxonomies have the same name but describe different aspects of a system, and even though most of the related taxonomies were created for program visualizations, the use of the term "program visualization" itself in certain contexts is described as "ambiguous" by Price et al. [60]. Therefore, we have reconciled the taxonomies described above into the SV portion of our Visualization Ontology. The benefit of using an ontology, rather than a taxonomy, is that we can show the relationships between the terms and concepts used by others in the field as well as the relationships between the different stages of the development of the system, the components of the system, and the usage of it.

6.3 Software Visualization Ontology Schema

The classes and properties of the visualization ontology schema consist of a reconciliation of existing taxonomies [9, 41, 52, 60, 69, 80] and additional features. Generally, the term "taxonomy" refers to a hierarchical structure [24] that represents "is a" relationships between levels. We have chosen to create an ontology because it permits us to specify the relationships between the categories and common terms, allows us to better show the stages of the development, animation, and viewing processes, which tend to be cyclic, and because an ontology permits the schema to grow in any direction. Ontologies are specified by a schema for a domain, in our case SV, and instances representing real world objects. The schema consists of classes with attributes and relationships to other classes.

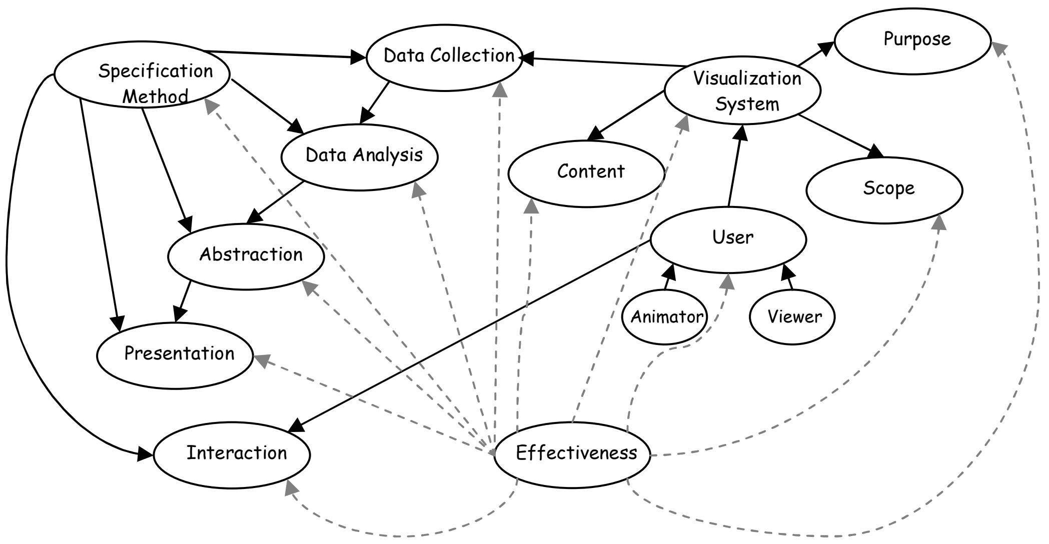


Figure 6.1: The Software Visualization Ontology Schema

Below are the classes of the software visualization portion of what will be an overall visualization ontology.

Software Visualization System

A software visualization system is used to create, modify, display, and interact with a software visualization. This class consists of attributes such as name, description, developer, url, institution, etc. that describe the background information of the system.

User

The animator and viewer, discussed above, are the users of the system. Both can be ranked as a novice, an intermediate, or an expert user.

Purpose

The reason for which an SV has been developed and the environment in which it is to be used helps to describe the purpose of an SV system. Some of the purposes that we have identified are for use in education, performance evaluation, debugging, testing, problem solving, software design, documentation, and empirical studies.

Content

This category classifies a system based on what it displays. A system may be designed to show code, data, algorithm or state of execution.

Scope

The scope of a software visualization system encompasses the range of programs that are viewable by the system as well as its portability. A system with a fixed scope can only show the visualizations created by the developer. A system with a scope that is not fixed provides more flexibility to the user.

Specification Method

As seen in Figure 6.1, specification method is directly related to the remaining classes. If the animator has control over the data that is collected or how it is analyzed and portrayed, then the specification method is the way in which she must modify the system to accomplish any one of these tasks. For example, a system may have a drag-and-drop style graphical user interface available for the animator to specify the type and location of graphics, or the animator may have to augment the source code to achieve the desired display.

Data Collection

This class addresses the questions of when, what, and how data is collected. If data is gathered, it could be collected at compile-time, run-time, or both. It might involve the collection of symbol table information, counts of low-level events, or the generation of higher-level "interesting events" through source code annotation. Event-based data collection stores data when an action by the user or another application occurs, whereas state-based data is collected as the status of a process changes.

Data Analysis

Once data is collected, some analysis may need to be done to obtain information of interest. If so, those calculations or transformations will take place during this stage. A very basic example would be determining the running time of a program. The data collected is usually the start-time and the stop-time, and a simple subtraction will compute the execution time.

Abstraction

Before data of interest are displayed, some systems allow the animator the capability to tailor the graphics to be specific to the data it represents. Visualizations that are very customized to the information it symbolizes are more abstract and may require more effort from the animator to specify than a generic display that the SV system can create automatically. Graphics are considered direct representations of data if they are nonspecific and there is a direct mapping from the graphic back to the data. Synthesized representations are created by the animator and attempt to depict his mental image of how the data should be displayed.

Presentation

Both hardware and software components are included in presentation. The display medium and use of audio could be supported by a color monitor and stereo speakers, respectively. Aspects of presentation controlled by the software include the use of color, dimension, and animation and the ability to show multiple views executing simultaneously.

Interaction

The interaction category describes the various ways that a viewer can interact with and control a visualization. The ability to zoom, control the speed and direction, and input data sets are common methods of interaction allowed to the viewer. This class also addresses how the viewer interacts (i.e. via a graphical interface or command line) with the system. Additionally, we have included a taxonomy created by Grissom et al. [27] that specifically addresses the viewer's level of interaction; the stages in increasing level of engagement are no viewing, viewing, responding, changing, constructing, and presenting.

Effectiveness

Effectiveness is arguably the most important class to users because they can utilize the ratings to decide if the system is worth investing the time needed to access and operate it. Developers may also find the effectiveness rating to be important. During the design phase, knowledge of some of the factors that influence the effectiveness of a system can help them to make better-informed decisions about what tools to use and how to use them. If the evaluation is of the developer's own system, the ratings can provide feedback on what others feel they did well or need to improve upon.

To ensure consistent ratings of the various systems, one person or a trained group could be responsible for using, evaluating and classifying each system. Since such a responsibility would require a tremendous amount of time and commitment, we are attempting to limit the subjectivity that can exist in rating systems by providing attributes that are clearly identifiable. The type for most of the properties is Boolean while other properties have fixed values that can be selected from a drop down list.

Our goal is to eventually be able to use basic characteristics of SVs and SV systems to compute an overall effectiveness rating. In the meantime, we will rely heavily on users' opinions and allow them to provide feedback about a system by rating the ease-of-use, the clarity to the user, the accuracy, and the overall rating. Two additional effectiveness properties that we currently include are the results of empirical studies, if they exist, and the acceptance of the system which is a measure of how long and how widely the system has been used.

6.4 VisIOn: Interactive Visualization Ontology

As stated earlier, the approach used to address the task of evaluating the effectiveness of software visualizations and systems starts with de-featuring them. For this task, the isolation of features is necessary for creating a method for classifying the systems. Attributes of the system and its capability of producing visualizations with specific features are identified as properties used to categorize the software visualization systems. These attributes and features have been enumerated as classes and properties of a visualization ontology schema. This ontology is interactive and available online through a tool called VisIOn (http://vision.cs.uga.edu).

Following in the same naming spirit as OWL (Web Ontology Language), VisIOn (Interactive Visualization Ontology) is a web application designed to categorize and store information about software visualization systems, in a way that can be easily searched and used for comparison. Information about each system can be entered in a detailed and impartial manner and as results of empirical studies continue to

As we and others continue to conduct empirical studies to investigate the contributions of individual features to the overall effectiveness of software visualizations, then the data entered into VisIOn can easily be used to provide an objective rating for each system.

6.4.1 Who will benefit?

The roles of the different people involved with an SV or SV system are an important consideration when creating this ontology. The *developer* is the person who designs and implements the SV system. The *animator* uses the system to create a visualization in hopes of conveying knowledge about the visualized program to the *viewer* who will watch and perhaps interact with the visualization and try to understand what the animator is attempting to portray.

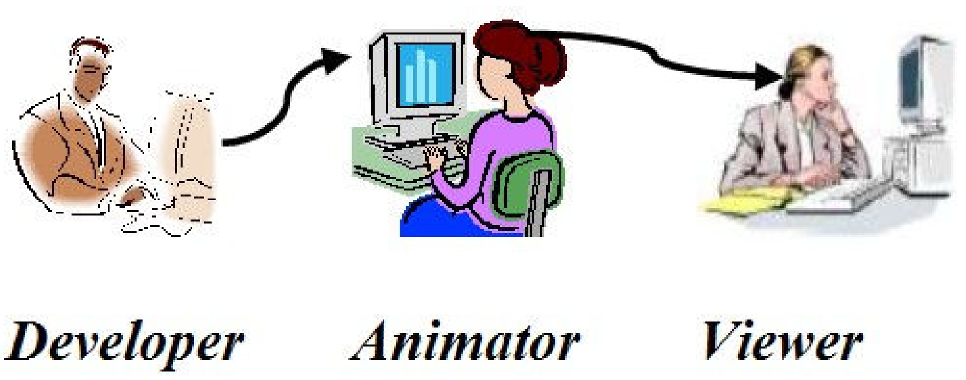


Figure 6.2: Software Visualization Users - A depiction of the roles of the people involved with an SV system or a visualization created using an SV system.

As illustrated in Figure 6.2, the developer's role is to design a system that gives the animator the tools to create an instance of a visualization with accompanying descriptions, interactions, and questions. The animator then uses the system to attempt to pass her knowledge to the viewer. VisIOn is intended to benefit people in each of these roles, as well as researchers, via its searching and comparing capabilities. Even though the same individual may perform multiple roles, the distinction is useful throughout the process of classifying and evaluating SV systems.

A **developer** in an environment such as academia, software design, or performance evaluation (*purpose*) decides what *content* is to be displayed and how an animator will be able to use and customize the SV (*specification method*) to obtain the range of desired visualizations (*scope*).

Despite the belief and desire for SVs to be useful pedagogical tools, the majority of professors of Computer Science courses do not use them, according to surveys by Naps et al. [55]. One common reason for this lack of use is the time required to find and/or create visualizations that will portray the examples and concepts of interest to the professor who, in this case, is playing the role of **animator**. VisIOn not only stores information about SV systems based on what they show and the system requirements to execute them, but it also provides information about the effectiveness of the SV including data on ease-of-use and acceptance (how many other people have used it and for how long). The animator can also find out what is required to make the system behave as she desires by viewing information stored under the *specification method* class. Another concern of professors is the lack of substantial empirical evidence to prove that SV systems enhance the quality of learning. Therefore, VisIOn allows its users to find systems that have been empirically evaluated, a list of the associated citation(s), and links to the papers where available.

A **viewer** of an SV system is usually trying to understand some aspect of a program. These users would want to quickly find a visualization that shows what they need and is appropriate for their level of expertise. VisIOn provides this information in the *content* and *effectiveness* classes.

Since VisIOn offers both a quick and an advanced search feature, a **researcher** who wants to find related systems and compare their functionalities can be very detailed in specifying what features they want to investigate or empirically evaluate. Once he has retrieved this initial list of systems, he can explore others that have been deemed "similar".

6.4.2 System Architecture

VisIOn was implemented using the Apache Struts framework, which encourages application architectures based on the Model-View-Controller (MVC) Model 2 design pattern [2]. MVC Model 2 design uses both JavaServer Pages (JSP) and servlets and allows communication between the view (JSP) and the model (JavaBean) through the controller (servlets) (See Figure 6.3).

The VisIOn *model* is stored in both a file and a relational database format and is accessed through JavaBeans. The file was created using Protégé-2000 [57], an integrated software tool used to develop knowledge-based systems. Protégé-2000 can create .owl files written in the Web Ontology Language (OWL), a standard format used for specifying an ontology schema. OWL is a vocabulary expressed using XML syntax that describes classes, their properties, and relationships between classes. Since querying a file is typically not as efficient as querying a relational database, the data from the .owl file is parsed once and stored into a MySQL database where data can be efficiently and easily retrieved.

The *view* is created using JSP pages that dynamically display content about the ontology schema and SV systems through the use of the JavaServer Pages Tag Library (JSTL) and a few Struts tag libraries. The data retrieved from the database are stored as Java Beans that can be directly accessed using JSP or indirectly accessed through the controller.

The *controller* consists of servlets that store and retrieve data, process user input, and determine flow-of-control. Based on a client's request, the controller forwards control to the appropriate view after supplying the necessary information to that JSP page. The org.apache.struts.action.ActionMapping configuration file (struts-config.xml) and classes that extend org.apache.struts.action.ActionForm and org.apache.struts.action.Action are considered components of the controller because they provide interaction between the model and the view and specify the forwarding actions. VisIOn uses ActionForms to store variables displayed and set via a JSP page. These variables may represent data from the model, or they may be values used to determine how data needs to be processed or which view is to be shown next. The Action classes contain the logic that uses these variables to create or retrieve a JavaBean and set the appropriate org.apache.struts.action.ActionForward. The struts-config.xml file, which contains a list of the ActionMappings, is then consulted to determine the action or webpage to which control is to be forwarded.

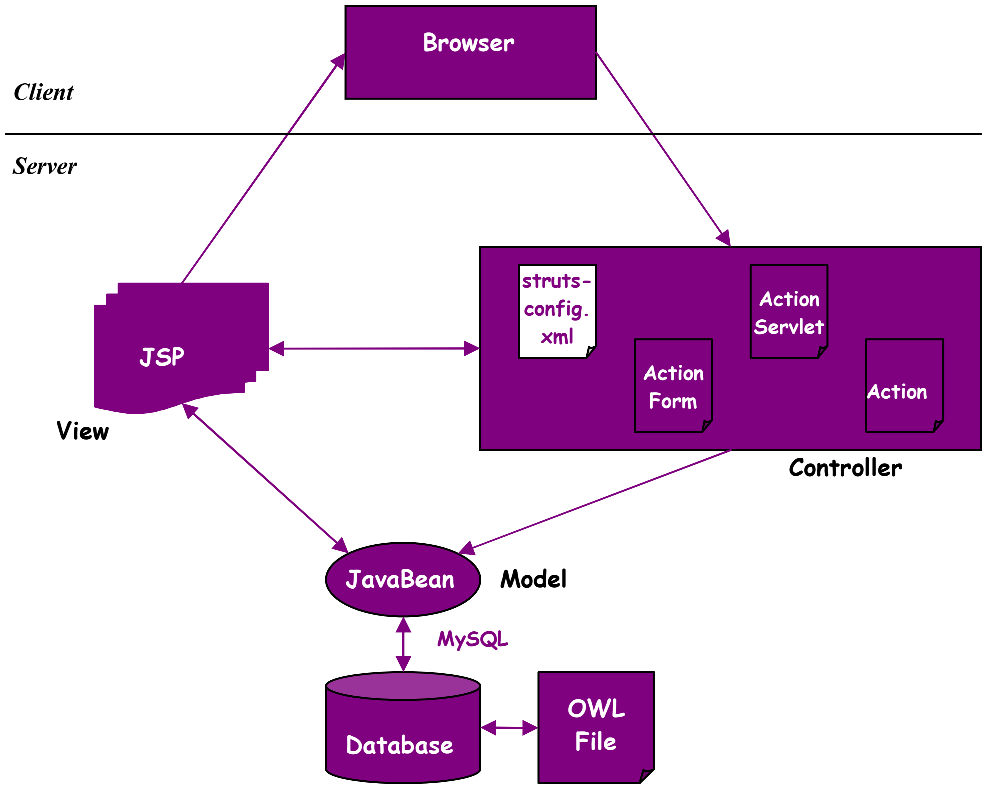
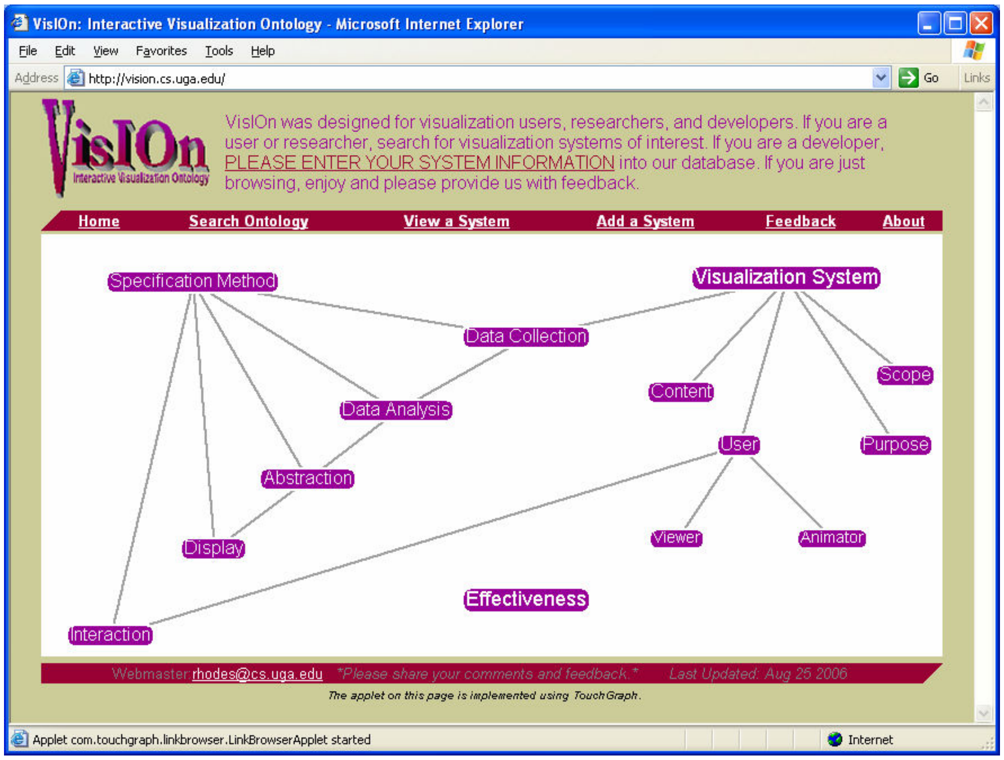


Figure 6.3: The VisIOn Architecture.

6.4.3 How to Use VisIOn

VisIOn provides developers, researchers, educators, and learners the ability to retrieve information about SV systems meeting specific criteria, to get detailed information about a particular system of interest, to add a system to the database, and to provide feedback about VisIOn or any SV system currently classified in VisIOn. Figure 6.4 is a screenshot of the homepage that displays a visualization of the SV ontology schema and a toolbar that allows access to the various VisIOn operations.



VisIOn was designed for visualization users, researchers, and developers. If you are a user or researcher, search for visualization systems of interest. If you are a developer, PLEASE ENTER YOUR SYSTEM INFORMATION into our database. If you are just browsing, enjoy and please provide us with feedback.

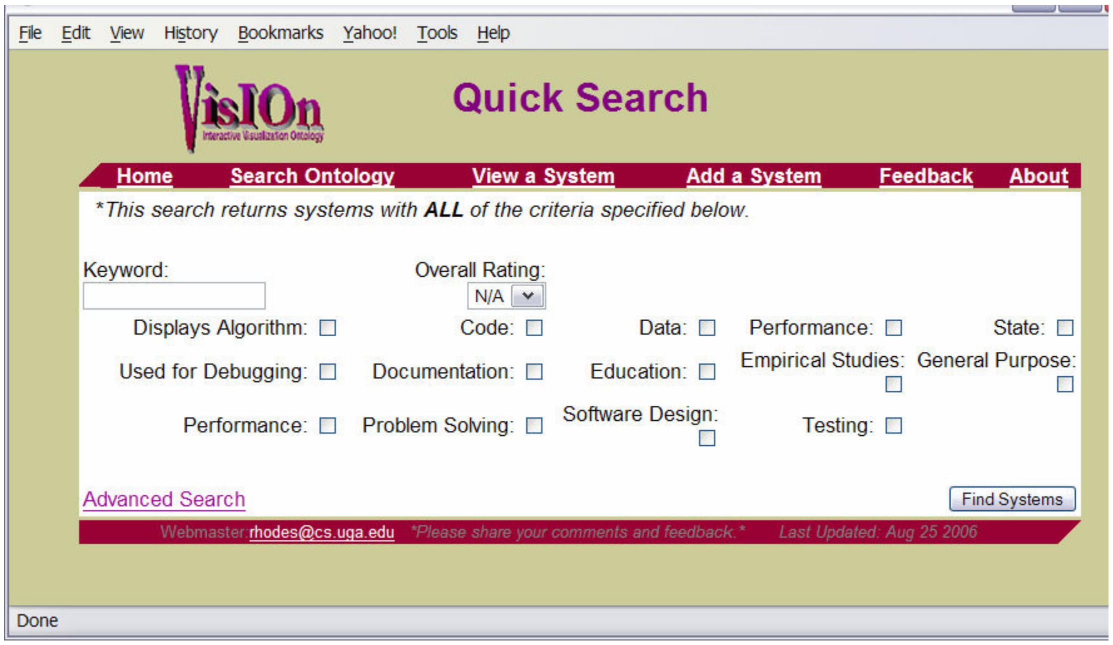
Webmaster rhodes@cs.uga.edu *Please share your comments and feedback.* Last Updated: Aug 25 2006

The applet on this page is implemented using TouchGraph.

Applet com.touchgraph.linkbrowser.LinkBrowserApplet started

Figure 6.4: The VisIOn Homepage.

The visualization on the homepage was created using TouchGraph [75], a set of interfaces for graph visualization that runs as an applet and requires an XML file as input. The nodes of the graph are the classes and the edges represent the relationships between the classes. Since *effectiveness* is related to all other classes, it would be connected to each of them and cause the display to be confusing, so currently, no edges are touching the *effectiveness* node. Hovering the cursor over a node will cause a screen to pop up that shows all of the properties for that class.



*This search returns systems with **ALL** of the criteria specified below.

Keyword:
Overall Rating: N/A

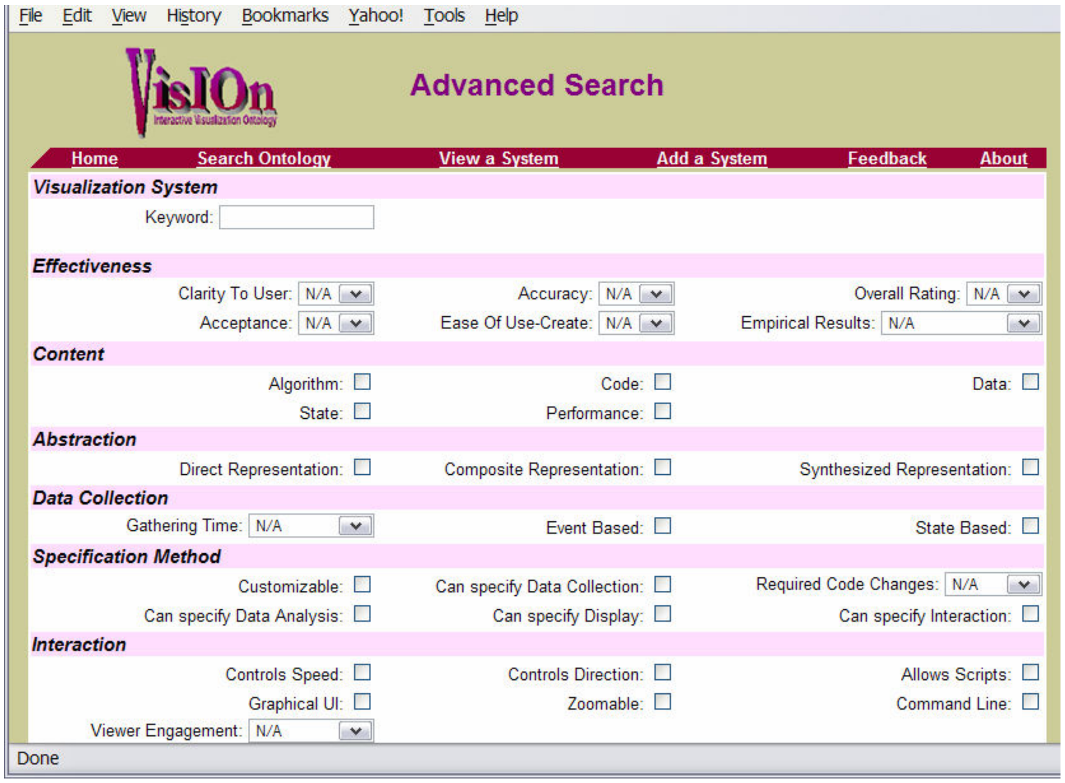
Displays Algorithm: ☐	Code: ☐	Data: ☐	Performance: ☐	State: ☐
Used for Debugging: ☐	Documentation: ☐	Education: ☐	Empirical Studies: ☐	General Purpose: ☐
Performance: ☐	Problem Solving: ☐	Software Design: ☐	Testing: ☐	

Advanced Search

Webmaster rhodes@cs.uga.edu *Please share your comments and feedback.* Last Updated: Aug 25 2006

Figure 6.5: The VisIOn Quick Search Page.

The first operation on the toolbar is “Search Ontology”. By clicking on this link, a user will be taken to a screen (see Figure 6.5) where he can search for systems that possess all of the selected characteristics of interest. The possible values for each property are most often true/false but also include predefined lists and open text. The “Keyword” field searches properties like the name of the SV system, developer, location, description, etc. that are not restricted to predefined values. Based on user feedback, the initial search screen was reduced to a simple search that only displays the properties that are thought to be of higher interest. For users who want to be very detailed in the type of SV systems they wish to retrieve, an advanced search is available by following the link at the bottom of the quick search screen.

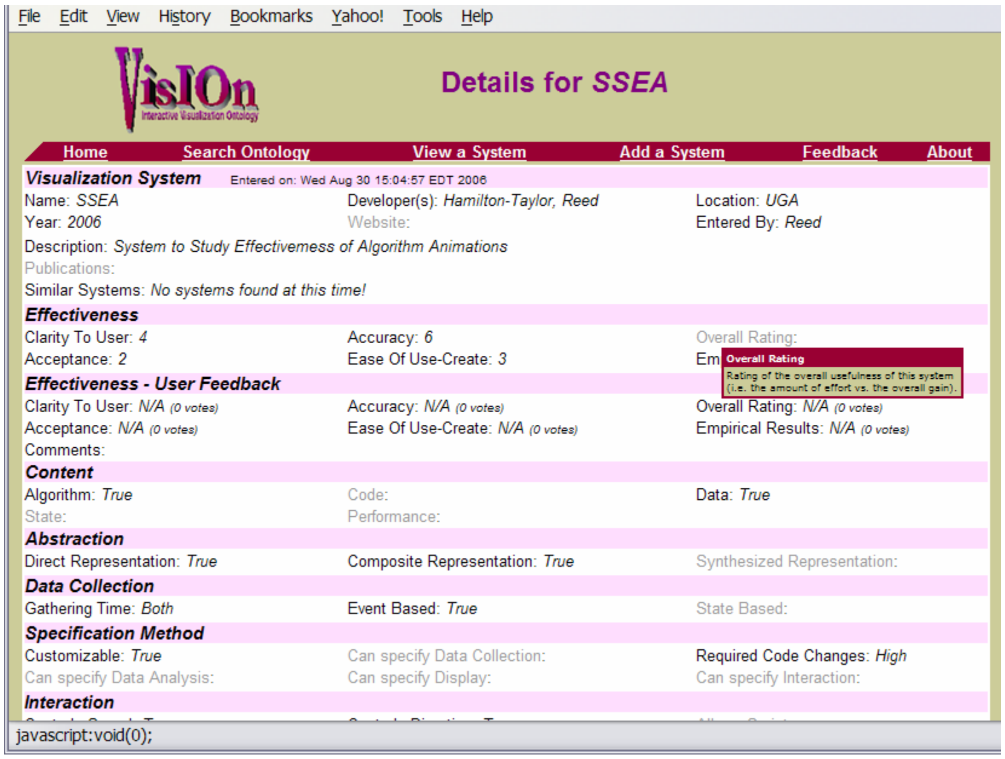

File Edit View History Bookmarks Yahoo! Tools Help
VisIOn
Interactive Visualization Ontology
Home
Search Ontology
View a System
Add a System
Feedback
About
Done

Advanced Search

Visualization System		
Keyword:		
Effectiveness		
Clarity To User: N/A	Accuracy: N/A	Overall Rating: N/A
Acceptance: N/A	Ease Of Use-Create: N/A	Empirical Results: N/A
Content		
Algorithm: ☐	Code: ☐	Data: ☐
State: ☐	Performance: ☐	
Abstraction		
Direct Representation: ☐	Composite Representation: ☐	Synthesized Representation: ☐
Data Collection		
Gathering Time: N/A	Event Based: ☐	State Based: ☐
Specification Method		
Customizable: ☐	Can specify Data Collection: ☐	Required Code Changes: N/A
Can specify Data Analysis: ☐	Can specify Display: ☐	Can specify Interaction: ☐
Interaction		
Controls Speed: ☐	Controls Direction: ☐	Allows Scripts: ☐
Graphical UI: ☐	Zoomable: ☐	Command Line: ☐
Viewer Engagement: N/A		

Figure 6.6: The VisIOn Advanced Search Page.

The Advanced Search screen (see Figure 6.6) lists each class followed by all properties with predefined values. The other properties are queried via the "Keyword" text field. Both the quick and advanced search screens display true/false values in the form of a checkbox for the user to check if they want systems that contain that property. Properties with predefined lists of values are shown in a drop-down menu for the user to select the one value of interest. The search screens are initialized with an empty "Keyword" field, unchecked boxes, and "N/A" selected for drop-down menus and will not be used in the search unless the user enters or selects a value of interest. After the user clicks on the "Find Systems" button, a list of system names meeting all of the search criteria is displayed. Each name is a link to a page with a list of properties specific to that system (see Figure 6.7).


File Edit View History Bookmarks Yahoo! Tools Help
VisIOn
Interactive Visualization Ontology
Home Search Ontology View a System Add a System Feedback About
javascript:void(0);

Details for *SSEA*

Visualization System Entered on: Wed Aug 30 15:04:57 EDT 2006

Name: *SSEA* Developer(s): *Hamilton-Taylor, Reed* Location: *UGA*
Year: *2006* Website: Entered By: *Reed*
Description: *System to Study Effectivemess of Algorithm Animations*
Publications:
Similar Systems: *No systems found at this time!*

Effectiveness

Clarity To User: *4* Accuracy: *6* Overall Rating:
Acceptance: *2* Ease Of Use-Create: *3* Em

Overall Rating
Rating of the overall usefulness of this system (i.e. the amount of effort vs. the overall gain).

Effectiveness - User Feedback

Clarity To User: *N/A (0 votes)* Accuracy: *N/A (0 votes)* Overall Rating: *N/A (0 votes)*
Acceptance: *N/A (0 votes)* Ease Of Use-Create: *N/A (0 votes)* Empirical Results: *N/A (0 votes)*
Comments:

Content

Algorithm: *True* Code: Data: *True*
State: Performance:

Abstraction

Direct Representation: *True* Composite Representation: *True* Synthesized Representation:

Data Collection

Gathering Time: *Both* Event Based: *True* State Based:

Specification Method

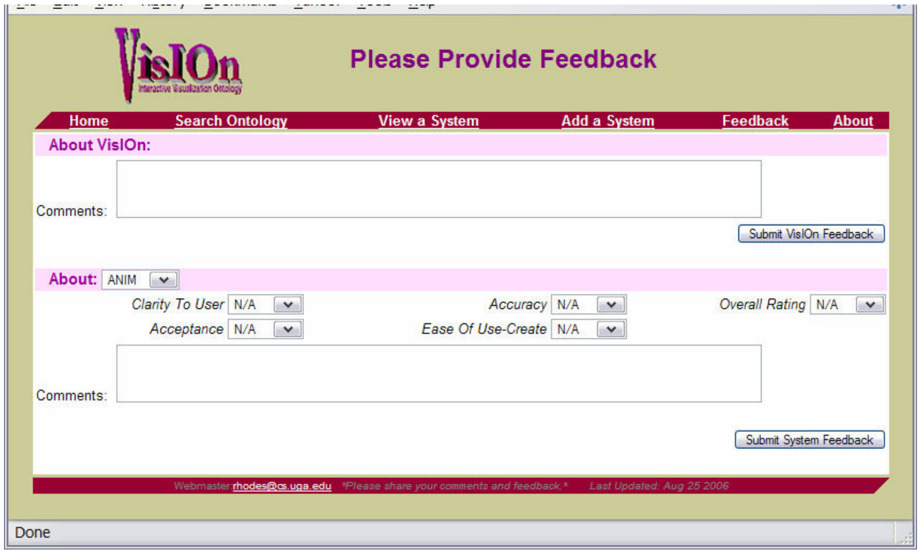
Customizable: *True* Can specify Data Collection: Required Code Changes: *High*
Can specify Data Analysis: Can specify Display: Can specify Interaction:

Interaction

Figure 6.7: The VisIOn System Properties Page.

The system properties page can also be reached by clicking on the "View a System" link on the toolbar, selecting the system of interest from a drop-down menu, and clicking on the "View Properties-List" button. Additionally, the properties and values of a system can be viewed on a page as a visualization that is identical to the SV ontology schema in Figure 6.4, but this time when the user hovers over a node, the pop-up screen will show properties for that class along with the specific values for each as they pertain to the selected system.

A potential user of VisIOn expressed an interest in seeing a list of systems similar to the one currently displayed. To provide users with such a listing, similarity between systems is currently computed using a *naïve* approach of giving each property a weight between 0 and 1 where all weights sum to 1. For each matching property value between two systems, the total match score is increased by the weight of the matched property. This approach will most likely improve to a collaborative filtering method in the future as we gather more user input and usage data.



Please Provide Feedback

About VisIOn:

Comments:

About: ANIM

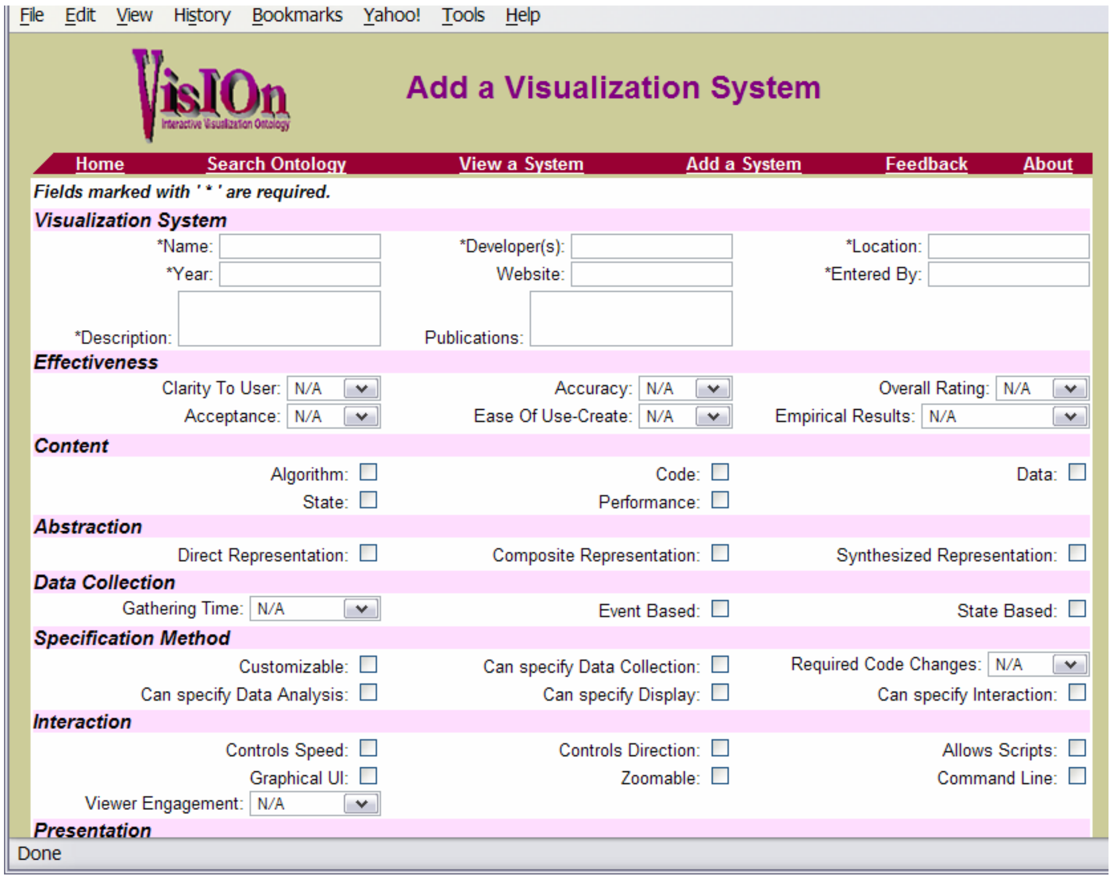
Clarity To User N/A
Accuracy N/A
Overall Rating N/A
Acceptance N/A
Ease Of Use-Create N/A

Comments:

Webmaster rhodes@cs.uga.edu *Please share your comments and feedback.* Last Updated: Aug 25 2006

Figure 6.8: The VisIOn Feedback Page.

The page displaying the system properties also includes a second *effectiveness* class that consists of user input supplied on the "Feedback" page, Figure 6.8. The *clarity to user*, *accuracy*, *ease-of-use*, and *overall rating* properties can currently be evaluated on a scale of 1 to 7, with 7 being the best, and the average of each property is displayed along with the number of voters. This type of rating or star system is common and helpful on websites that provide access to products.


File Edit View History Bookmarks Yahoo! Tools Help
VisIOn
Interactive Visualization Ontology
Home
Search Ontology
View a System
Add a System
Feedback
About
Done

Add a Visualization System

*Fields marked with ' * ' are required.*

Visualization System

*Name:		*Developer(s):		*Location:	
*Year:		Website:		*Entered By:	
*Description:		Publications:			

Effectiveness

Clarity To User:	N/A	Accuracy:	N/A	Overall Rating:	N/A
Acceptance:	N/A	Ease Of Use-Create:	N/A	Empirical Results:	N/A

Content

Algorithm:	☐	Code:	☐	Data:	☐
State:	☐	Performance:	☐		

Abstraction

Direct Representation:	☐	Composite Representation:	☐	Synthesized Representation:	☐

Data Collection

Gathering Time:	N/A	Event Based:	☐	State Based:	☐

Specification Method

Customizable:	☐	Can specify Data Collection:	☐	Required Code Changes:	N/A
Can specify Data Analysis:	☐	Can specify Display:	☐	Can specify Interaction:	☐

Interaction

Controls Speed:	☐	Controls Direction:	☐	Allows Scripts:	☐
Graphical UI:	☐	Zoomable:	☐	Command Line:	☐
Viewer Engagement:	N/A				

Presentation

Figure 6.9: The VisIOn "Add a System" Page.

The last function provided to users is the ability to "Add a System" to the VisIOn database (see Figure 6.9). The SV system *Name*, *Developer*, *Description*, and the name of the person entering the system (*Your Name*) are mandatory fields and they, along with a minimum of any 4 other properties, must be entered before a system is accepted into the repository. Upon insertion into the database, the information about the SV system is instantly available and similar systems are computed and stored. The newly added systems will be verified regularly in order to maintain a level of quality that will be beneficial to our users.

6.5 Vision for VisIOn

We will continue to develop, refine, and promote VisIOn in hopes of others adding accurate information about software visualization systems so that our goal of making this data easily accessible may be achieved. We will frequently monitor the quality of information entered due to the unrestricted ability of users to add and rate systems. We may have to allow this function on a more selective basis if users enter information other than in a good faith manner. As we continue to categorize systems and more users provide feedback, we will possibly update classes, properties, and available functionality.

Although a visualization can be classified using VisIOn, it is currently set up to house information about software visualization systems. To categorize a particular visualization, consideration will need to be given as to how it will be connected to the system that was used to create it, if one was used. Currently, any information about the visualization systems that is significant but not requested by one of the properties can be supplied in the description.

We hope that the results of software visualization related studies will lead to more objective criteria for evaluating visualizations and systems and that we will be able to apply what is learned to other visualization domains. If we find that specific features commonly increase the level and/or speed of understanding, then we can start to evaluate visualizations and systems in other fields and expand VisIOn to include information about them. Please feel free to access VisIOn at http://vision.cs.uga.edu and enter data or provide feedback.

CHAPTER 7

CONCLUSION

7.1 Summative Analysis

Chapters 4 and 5 describe a series of studies conducted to investigate the effects of several perceptual, attentional, and cognitive features commonly employed in software visualizations, specifically those used in educational settings by students learning about algorithms. We started by describing two low-level, perceptual/attentional studies performed using context-free software packages, *JND* and *VizEval*. The generic findings from these studies pertaining to bar height differences and the use of a flashing cue in detection and localization of changes can be applied within any visualization domain in which graphical objects are rectangular and change size.

We integrated knowledge gained from these low-level studies into higher-level, domain-specific studies in which we investigated the influence of different features on perception of an animation and overall comprehension of the depicted algorithm. These higher-level studies investigated the impact of certain types of cueing, motion, and interactive questioning techniques on the effectiveness of AAs and AA systems. Although participants in these studies were specifically targeted to have similar educational backgrounds, we are aware of their varying memory, reasoning, and spatial capabilities as well as their differing learning styles. Experiments involving human subjects, especially at this level of cognitive reasoning, inevitably are hindered by this obstacle, unlike studies in other areas that involve machines built to a standard specification or chemicals composed of precisely mixed elements. Starting to identify differences in performance and attention based on these individual differences will give researchers in the software visualization domain another perspective from which to design systems.

7.2 Limitations

Three possible limitations that we have identified with the AA studies are that:

1) Features are only tested on one algorithm, quicksort. Some of the features will likely have different results for algorithms with various levels of complexity,

2) The motivation for students to perform well is completely intrinsic because the results are anonymous and will not count towards a grade for a course, and

3) AA systems are typically intended to be used as an additional resource. So, students may attend a lecture and then use or create an AA. The format for our experiments involved students learning the algorithm solely through the use of the animation *(not ethnographic)*.

7.3 Conclusion and Future Work

Well designed AAs and AA systems stand to offer great pedagogical benefits because they have the potential to enhance a student's understanding with less time invested than traditional, non-interactive, static descriptions and depictions. The fact that some studies have proven this to be true while others have not has motivated this research and the desire to find out "*Why?*". In this document, a methodology has been proposed that is intended to be useful in successfully designing and evaluating the effectiveness of AAs and AA systems. This research has just begun to scratch the surface of this well acknowledged and studied problem. Hopefully, the outcome of this and other related research will lead to the successful and consistent creation of SVs and SV systems that will enhance the computer science educational environment.

As we continue to identify perceptual, attentional, and cognitive factors of visualizations that may influence their effectiveness, we will perform empirical studies to validate (or disprove) our hypotheses. Hopefully, our work in conjunction with that of others in our field will help to remove some of the subjectivity involved with evaluating the effectiveness of visualizations. Once effective AAs can be described and implemented, perhaps professors will be more accepting of them, incorporate them into computer science courses, and afford students the opportunity to benefit from them.

The increase in the attendance at and number of software visualization related workshops indicates that this field has much potential for future research. In these settings, I have found that other researchers are very interested in formally testing the usefulness of their tools, but are not sure how to proceed. I believe that the software and evaluation process that I am describing in this work can serve as a framework for others to use or follow. The algorithm animation community stands to greatly benefit from additional studies on the perceptual, attentional, and cognitive features of these visualizations.

Also, we have started to screen students for individual differences in areas such as working memory, inductive reasoning, learning styles, viewing patterns, etc. Future work in this area could involve creating intelligent software visualization systems that are capable of tailoring views and interaction based upon the *type* of user.

During my doctoral research, I have also worked with a bioinformatics project where my contribution involved enhancing the usability of web sites and the visualizations used. An interesting idea is to explore the possibility of extending our software visualization findings into the new and neglected area of bioinformatics visualization. There appears to be a great need for an approach similar to the one that we have developed for systematically, quantitatively designing and evaluating bioinformatics visualizations since effective ones will assist biologists in interpreting data that could lead to medical and other scientific breakthroughs.

REFERENCES

[1] M. Agarwal. Viewing Behavior Model Graphs (VBMG) for Characterizing User Viewing Behavior in Program Visualizations. Unpublished MS Thesis, The University of Georgia, Athens, GA, 2007

[2] Apache Struts [Online]. Available at <http://struts.apache.org/>.

[3] A. Badre, M. Beranek, J. M. Morris, and J. Stasko. Assessing Program Visualization Systems as Instructional Aids. In: *Proceedings of Computer Assisted Learning, ICCAL '92*, Wolfville, Nova Scotia, Canada, 1992.

[4] R. Baecker. Sorting out Sorting, 30 minute color/sound film, University of Toronto. Distributed by Morgan Kaufmann, San Francisco. 1981.

[5] R. Baecker. Sorting out sorting: A case study of software visualization for teaching computer science. Chapter in: *Software Visualization: Programming as a Multimedia Experience,* edited by J. D. M. Brown, B. Price, and J. Stasko, Cambridge, MA: The MIT Press, pp. 369-381, 1998.

[6] K. Borner and Y. Zhou. A Software Repository for Education and Research in Information Visualization. In: *Proceedings of Fifth International Conference on Information Visualisation (IV '01)*, London, England, p. 257, 2001.

[7] Encyclopaedia Britannica Online [Online]. Available at <http://www.britannica.com/>.

[8] K. W. Brodlie, D. A. Duce, D. J. Duke, and et al. Visualization Ontologies: Report of a Workshop held at the National e-Science Centre. Technical Report, e-Science Institute, Edinburgh, Scotland, 2004.

[9] M. H. Brown. Perspectives on algorithm animation. In: *Proceedings of ACM SIGCHI '88 Conference on Human Factors in Computing Systems*, Washington, DC, pp. 33-38, 1988.

[10] M. H. Brown and J. Hershberger. Color and Sound in Algorithm Animation. *j-COMPUTER* vol. 25, pp. 52-63, 1991.

[11] M. H. Brown and S. Robert. A system for algorithm animation. *ACM Computer Graphics,* vol. 18, pp. 177-186, 1984.

[12] Complete Collection of Algorithm Animations [Online]. Available at <http://www.cs.hope.edu/~alganim/ccaa/>.

[13] M. D. Byrne, R. Catrambone, and J. T. Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers & Education* vol. 33, pp. 253-278, 1999.

[14] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. San Francisco, CA, 1999.

[15] J. Carroll. *Human-Computer Interaction in the New Millennium*: Addison Wesley Pub Co Inc, 2001.

[16] Computer Science Teaching Center [Online]. Available at <http://www.cstc.org/index.html>.

[17] P. Crescenzi, N. Faltin, R. Fleischer, C. Hundhausen, S. Näher, G. Rößling, J. Stasko, and E. Sutinen. The Algorithm Animation Repository. In: *Proceedings of Second International Program Visualization Workshop*, HornstrupCentret, Denmark pp. 14-16, 2002.

[18] E. Dale. *Audio-Visual Methods in Teaching*. New York, NY: Dryden Press, 1954.

[19] E. T. Davis, K. Hailston, E. Kraemer, A. Hamilton-Taylor, P. Rhodes, C. Papadimitiou, and B. Garcia. Examining Perceptual Processing of Program Visualization Displays to Enhance Effectiveness. In: *Proceedings of Human Factors and Ergonomics Society (HFES)*, 2006.

[20] American Heritage Dictionary [Online]. Available at <http://education.yahoo.com/reference/dictionary/>.

[21] D. J. Duke, K. W. Brodlie, and D. A. Duce. Building an Ontology of Visualization. in *Proceedings of the Conference on Visualization '04*: IEEE Computer Society, 2004.

[22] D. J. Duke, K. W. Brodlie, D. A. Duce, and I. Herman. Do You See What I Mean? *IEEE Computer Graphics and Applications* vol. 25, pp. 6 - 9, 2005.

[23] R. B. Ekstrom, J. W. French, H. H. Harman, and D. Derman. *The Kit of Factor-Referenced Cognitive Tests*. Princeton, NJ: Educational Testing Service (ETS), 1976.

[24] WIKIPEDIA: The Free Encyclopedia, Definition of "Taxonomy" [Online]. Available at <http://en.wikipedia.org/wiki/Taxonomy>.

[25] R. M. Felder and L. K. Silverman. Learning and Teaching Styles in Engineering Education. *Journal of Engineering Education,* vol. 78, pp. 674-681, 1988.

[26] R. Fleischer. COMP272: Theory of Computing - A study on the learning effectiveness of visualizations. In: *Proceedings of Second Teaching and Learning Symposium - Teaching Innovations: Continuous Learning and Improvement*, Hong Kong, 2004.

[27] S. Grissom, M. F. McNally, and T. Naps. Algorithm visualization in CS education: comparing levels of student engagement. In: *Proceedings of the 2003 ACM symposium on Software visualization,* San Diego, California: ACM Press, 2003.

[28] J. S. Gurka and W. Citrin. Testing effectiveness of algorithm animation. In: *Proceedings of IEEE Symposium on Visual Languages*, Los Alamitos, CA, pp. 182-189, 1996.

[29] A. G. Hamilton-Taylor and E. Kraemer. SKA: Supporting Algorithm and Data Structure Discussion. In: *Proceedings of 33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE '02)*, Cincinnati, Kentucky, pp. 58-62, 2002.

[30] S. Hansen, N. H. Narayanan, and M. Hegarty. Designing educationally effective algorithm visualizations. *Journal Of Visual Languages And Computing,* vol. 13, pp. 291-317, 2002.

[31] A. Hausner and D. P. Dobkin. GAWAIN: Visualizing Geometric Algorithms with Web-based Animation. In *Proceedings of the fourteenth annual symposium on Computational geometry,* Minneapolis, Minnesota, United States: ACM Press, 1998.

[32] C. G. Healy. Building a Perceptual Visualization Architecture. *Behaviour and Information Technology,* vol. 19, pp. 349-366, 2000.

[33] C. G. Healy. On the Use of Perceptual Cues and Data Mining for Effective Visualization of Scientific Datasets. In: *Proceedings of Graphics Interface*, Vancouver, Canada, pp. 177-184, 1998.

[34] C. G. Healy and J. T. Enns. Large Datasets at a Glance: Combining Textures and Colors in Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics,* vol. 5, 1999.

[35] T. Hübscher-Younger and N. H. Narayanan. Dancing Hamsters and Marble Statues: Characterizing Student Visualizations of Algorithms. In: *Proceedings of ACM Symposium on Software Visualization*, 2003.

[36] C. D. Hundhausen and S. A. Douglas. Using visualizations to learn algorithms: Should students construct their own, or view an expert's?. In: *Proceedings of IEEE International Symposium on Visual Languages*, pp. 21-28, 2000.

[37] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A Meta-Study of Algorithm Visualization Effectiveness. *Journal Of Visual Languages And Computing,* vol. 13, pp. 259-290, 2002.

[38] D. Jarc, M. Feldman, and R. Heller. Assessing the benefits of interactive prediction using Web-based algorithm animation courseware. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education,* Austin, Texas, United States: ACM Press, 2000.

[39] D. J. Jarc and M. B. Feldman. An Empirical Study of Web-based Algorithm Animation Courseware in an Ada Data Structure Course. In: *Proceedings of ACM SIGAda international conference on Ada* Washington, DC, 1998.

[40] S. Kaldate. Analysis of Viewing Behavior of Program Visualization and Interaction with Individual Differences. Unpublished MS Thesis, The University of Georgia, Athens, GA, 2007

[41] E. Kraemer and J. T. Stasko. The Visualization of Parallel Systems: An Overview. *Journal of Parallel and Distributed Computing,* vol. 18, pp. 105-117, 1993.

[42] E. T. Kraemer, B. Reed, P. Rhodes, and A. Taylor. SSEA: A System for Studying the Effectiveness of Animations. In: *Proceedings of Fourth Program Visualization Workshop (PVW2006)*, University of Florence, Italy, 2006.

[43] Weber's Law and Fechner's Law *(class handout)* [Online]. Available at <http://www.cns.nyu.edu/~msl/courses/0044/handouts/Weber.pdf>.

[44] A. Lawrence. Empirical studies of the value of algorithm animation in algorithm understanding. Unpublished PhD Dissertation, Georgia Institute of Technology, Atlanta, GA, 1993

[45] A. L. Lawrence, A. N. Badre, and J. T. Stasko. Empirically Evaluating the use of Animations to Teach Algorithms. In: *Proceedings of IEEE Symposium on Visual Languages* pp. 48-54, 1994.

[46] LookWAYup [Online]. Available at <http://lookwayup.com/free/>.

[47] J. D. MacKinlay. Automating the design of graphical presentation of relational information. *ACM Transaction on Graphics,* vol. 5, pp. 110-141, 1986.

[48] Human Factors [Online]. Available at <http://www.sis.pitt.edu/~mariah/spring2002/is1052/human_factors.htm>.

[49] R. E. Mayer and R. Moreno. Nine Ways to Reduce Cognitive Load in Multimedia Learning. *EDUCATIONAL PSYCHOLOGIST,* vol. 38, pp. 43 - 52, 2003.

[50] MedTerms Medical Dictionary [Online]. Available at <http://www.medterms.com/script/main/hp.asp>.

[51] E. Morse, M. Lewis, and K. A. Olsen. Evaluating Visualizations: Using a Taxonomic Guide. *International Journal of Human Computer Systems,* vol. 53, pp. 637-662, 2000.

[52] B. A. Myers. Taxonomies of Visual Programming and Program Visualization. *Journal of Visual Languages and Computing,* vol. 1, pp. 97-123, 1990.

[53] T. Naps, J. Eagan, and L. Norton. JHAVE: An Environment to Actively Engage Students in Web-Based Algorithm Visualization. In: *Proceedings of Symposium on Computer Science Education (SIGCSE 2000)*, Austin, TX, pp. 109-113, 2000.

[54] Improving the Educational Impact of Algorithm Visualization [Online]. Available at <http://www.algoanim.net/>.

[55] T. L. Naps, G. Rößling , V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and V. J. Angel, Iturbide. Exploring the role of visualization and engagement in computer science education. In *Working group reports from ITiCSE on Innovation and technology in computer science education,* Aarhus, Denmark: ACM Press, 2002.

[56] National Training Laboratories in Bethel, Maine [Online]. Available at < http://www.ntl.org >.

[57] N. F. Noy, R. W. Fergerson, and M. A. Musen. The knowledge model of Protege-2000: Combining interoperability and flexibility. In: *Proceedings of 2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000)*, Juan-les-Pins, France, 2000.

[58] F. Paas, J. E. Tuovinen, H. Tabbers, and P. W. M. V. Gerven. Cognitive load measurement as a means to advance cognitive load theory. *EDUCATIONAL PSYCHOLOGIST,* vol. 38, pp. 63 - 71, 2003.

[59] J. Preece, Y. Rogers, and e. a. Helen Sharp. *Human-Computer Interaction*. Reading, MA: Addison-Wesley, 1994.

[60] B. A. Price, R. M. Baecker, and I. S. Small. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing,* vol. 4, pp. 211-266, 1993.

[61] W. Prinzmetal, A. Zvinyatskovskiy, and L. Dilem. Voluntary and Involuntary Attention Have Different Consequences: The Effect of Perceptual Difficulty. Submitted for publication.

[62] Dept. of Psychology, Wright State University [Online]. Available at <http://www.psych.wright.edu/HFHOME.HTM>.

[63] B. Reed. Investigating Characteristics of Effective Program Visualizations: A Testing Environment and the Effect of Cueing and Swapping Techniques in Algorithm Animations. Unpublished MS Thesis, University of Georgia, Athens, GA, 2006

[64] B. Reed, P. Rhodes, E. Kraemer, A. Hamilton-Taylor, E. T. Davis, and K. Hailston. The Effect of Comparison Cueing and Exchange Motion on Comprehension of Program Visualizations. In: *Proceedings of ACM Symposium on Software Visualization 2006 (SOFTVIS'06)*, Brighton, UK, 2006.

[65] P. Rhodes, E. Kraemer, A. Hamilton-Taylor, S. Thomas, M. Ross, E. Davis, K. Hailston, and K. Main. VizEval: An Experimental System for the Study of Program Visualization Quality. In: *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) 2006*, Brighton, UK, 2006.

[66] P. Rhodes, E. Kraemer, and B. Reed. The Importance of Interactive Questioning Techniques in the Comprehension of Software Visualizations. In: *Proceedings of ACM Symposium on Software Visualization 2006 (SOFTVIS'06)*, Brighton, UK, 2006.

[67] P. Rhodes, E. Kraemer, and B. Reed. VisIOn: An Interactive Visualization Ontology. In: *Proceedings of 44th ACM Southeast Conference (ACM SE '06)*, Melbourne, FL, 2006.

[68] T.-M. Rhyne, T. H. D. Jr., G. Calapristi, C. North, and D. Gresh. Evolving Visual Metaphors and Dynamic Tools for Bioinformatics Visualization. In: *Proceedings of IEEE Visualization*, Boston, MA, 2002.

[69] G. C. Roman and K. C. Cox. A Taxonomy of Program Visualization Systems. *IEEE Computer,* vol. 26, pp. 11-24, 1993.

[70] M. Ross. A testing environment for the evaluation of program visualization quality. Unpublished MS Thesis, The University of Georgia, Athens, GA, 2004

[71] G. Rößling and T. L. Naps. A Testbed for Pedagogical Requirements in Algorithm Visualizations. In: *Proceedings of 7th Annual ACM SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002)*, Arhus, Denmark, 2002.

[72] P. Saraiya, C. North, and K. Duca. Visualizing biological pathways: requirements analysis, systems evaluation and research agenda. *Information Visualization,* vol. 4, pp. 191-205, 2005.

[73] P. Saraiya, C. A. Shaffer, D. S. McCrickard, and C. North. Effective Features of Algorithm Visualizations. In: *Proceedings of 2004 ACM Technical Symposium on Computer Science Education (SIGCSE)*, Norfolk, VA, 2004.

[74] M. Scaife and Y. Rogers. External Cognition: How Do Graphical Representations Work?. *International Journal of Human-Computer Studies,* vol. 45, pp. 185-213, 1996.

[75] TouchGraph LLC [Online]. Available at <http://www.touchgraph.com/index.html>.

[76] J. Stasko. POLKA animation designer's package. Technical Report, Georgia Institute of Technology, Atlanta, GA, 1995.

[77] J. T. Stasko. Animating algorithms with XTANGO. *SIGACT News,* vol. 23, pp. 67-71, 1992.

[78] J. T. Stasko. TANGO: A framework and system for algorithm animation. *IEEE Computer,* vol. 23, pp. 27-39, 1990.

[79] J. T. Stasko. Using Student-Built Animations As Learning Aids. In: *Proceedings of the ACM Technical Symposium on Computer Science Education,* pp. 25-29, 1997.

[80] J. T. Stasko and C. Patterson. Understanding and Characterizing Program Visualization Systems. In: *Proceedings of IEEE Symposium on Visual Languages*, pp. 3-10, 1992.

[81] L. Stern, S. Markham, and R. Hanewald. You can lead a horse to water: how students really use pedagogical software. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education,* Caparica, Portugal: ACM Press, 2005.

[82] B. M. Sullivan, C. Ware, and M. Plumlee. Linking audio and visual information while navigating in a virtual reality kiosk display. *Journal of Educational Multimedia and Hypermedia,* in press.

[83] Math Archives [Online]. Available at <http://archives.math.utk.edu/>.

[84] S. Thomas. An experiment designer tool for evaluation of program visualization quality. Unpublished MS Thesis, The University of Georgia, Athens, GA, 2004

[85] M. E. Tudoreanu. Designing Effective Program Visualization Tools for Reducing User's Cognitive Effort. In: *Proceedings of ACM Symposium on Software Visualization SOFTVIS '03*, San Diego, CA, 2003.

[86] R. e. a. Turner. Visualization Challenges for a New Cyberpharmaceutical Computing Paradigm. In: *Proceedings of IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, pp. 7-18, 2001.

[87] B. Tversky. Animation: Can it Facilitate?. *International Journal of Human-Computer Studies,* vol. 57, pp. 247-262, 2002.

[88] Netlib Repository at UTK and ORNL [Online]. Available at <http://www.netlib.org>.

[89] WIKIPEDIA: The Free Encyclopedia [Online]. Available at <http://en.wikipedia.org/wiki/Taxonomy>.

APPENDIX A

SAMPLE CONSENT FORM AND HANDOUT

CONSENT FORM

I, ____, agree to participate in a research study titled "The Importance of Pop-up Questioning Techniques in Comprehension of Program Visualizations" conducted by Philippa Rhodes from the Department of Computer Science at the University of Georgia (706-XXX-XXXX) under the direction of Dr. Eileen Kraemer, Department of Computer Science, University of Georgia (706-XXX-XXXX). I understand that my participation is voluntary. I can stop taking part without giving any reason, and without penalty. I can ask to have all of the information about me returned to me, removed from the research records, or destroyed.

The reason for this study is to evaluate the importance of various attributes in algorithm animations.

If I volunteer to take part in this study, I will be asked to do the following things:
1) Read introductory material, including a short demo which will last approximately 15 minutes.
2) Use the SSEA program to learn and answer questions about the quicksort algorithm which will take up to 45 minutes.

I will learn how the quicksort algorithm works, when to use this sorting procedure, and how well this algorithm performs on various inputs.

I will receive $10.00 at the completion of my participation of this study.

No risks are expected. No discomforts or stresses are expected.

No individually identifying information about me, or provided by me during the research, will be shared with others without my written permission, except if required by law. The results of this participation will be confidential.

The researcher will answer any further questions about the research, now or during the course of the project, and can be reached by telephone at: 706-XXX-XXXX.

I understand the procedures described above. My questions have been answered to my satisfaction, and I agree to participate in this study. I have been given a copy of this form.

Philippa Rhodes		
Name of Researcher	Signature	Date
Telephone: 706-XXX-XXXX		
Email: rhodes@cs.uga.edu		
Name of Participant	**Signature**	**Date**

Please sign both copies. You are to keep one and return one to the researcher.

Additional questions or problems regarding your rights as a research participant should be addressed to The Chairperson, Institutional Review Board , University of Georgia, 612 Boyd Graduate Studies Research Center, Athens, Georgia 30602-7411; Telephone (706) XXX-XXXX; E-Mail Address IRB@uga.edu

Instructions for this Algorithm Animation Study

Before you begin

- Please read and sign one consent form. The other copy is for you to keep.
- If you have any questions, please ask the researcher at this time.
- We ask that you:
    - Please turn your mobile phone off or to a silent ring.
    - Please do not run any other computer programs during the experiment.
    - Do not write your name on this or any other sheet.

What you will be doing

You will view two algorithm animations and answer questions in a program called SSEA (System to Study Effectiveness of Animations). The purpose of the first animation is to allow you to practice using the SSEA program. The purpose of the second animation is for you to learn about the quicksort algorithm. For the quicksort portion of this study, you will be asked to:

1. complete a questionnaire,
2. complete a pre-test
3. view the animation until you are comfortable with this quicksort algorithm
4. complete a post-test
5. complete a series of preference and abilities tests (e.g., acuity screening, color vision, spatial ability, etc).

Also included with this handout are:

- a SSEA Guide, "How to Use SSEA",
- scratch paper, and
- a feedback form.

Running the Demo

The demo depicts an animation of a simple algorithm that finds the maximum value in an array. You will be asked to run the animation on two data sets and to then answer questions about the algorithm.

1. Double click on the "SSEA_Demo" icon on the computer's desktop.
2. Click on **Begin Animation** for "**Demo1**" and watch the animation (Use ② in Figure B of "How to Use SSEA"). It will run to completion. You may click the **Begin Animation** button again if you wish to see the animation again.
3. Change input to **"Demo2"** (Use ① in Figure B). Select **Yes** when asked "Are you sure you wish to change the inputs?". Then follow steps 4 – 9 below.

4. Click on **Begin Animation**. An animation of the algorithm as it executes on a new data set will begin to play.
5. Click on **Pause** (③) to pause the animation.
6. Click the **Step** (④) button multiple times to advance the animation step-by-step. Stop at step 10. If necessary, press **Begin Animation** to restart the animation.
7. Press **Play** (③) to return to continuous play mode.
8. Move the "**Step" slider** (⑥) back to demo step 0. Reply **Yes** when asked if you want to restart the animation at the selected step.
9. Move the **"Speed" slider** (⑤) to Faster.
10. Now look at the question area in the lower left corner (Area 3 in Figure A). Answer **Question 1**. You may replay the animations to help you answer the questions.
11. Select **Question 2** by clicking on the "Question 2" label in the leftmost panel or using the **Next** button in the lower left corner.
12. Select and then answer **Question 3** and **Question 4**. In addition to replaying the animations, you may also view the code panel on the right-hand side of the display (Area 2 in Figure A).
13. You may go back and change answers to previous questions if desired. Once you are done answering all the questions, click **Submit**.
14. Close down the SSEA_DEMO windows.

Do you have any questions? Please ask a researcher.

Running the Quicksort SSEA:

1. Double click on the "SSEA" icon on the computer's desktop.
2. Use the animations and the various data sets, as well as the code display to answer all questions about this version of the quicksort algorithm.
3. Feel free to pause, step, replay, and adjust the speed as necessary to help you answer the questions.
4. When you are done, click **Submit** and close the SSEA windows.

Before you leave, please give us any comments on the feedback form (attached).

Thank you for your time!

How to Use SSEA
(System to Study Effectiveness of Animations)

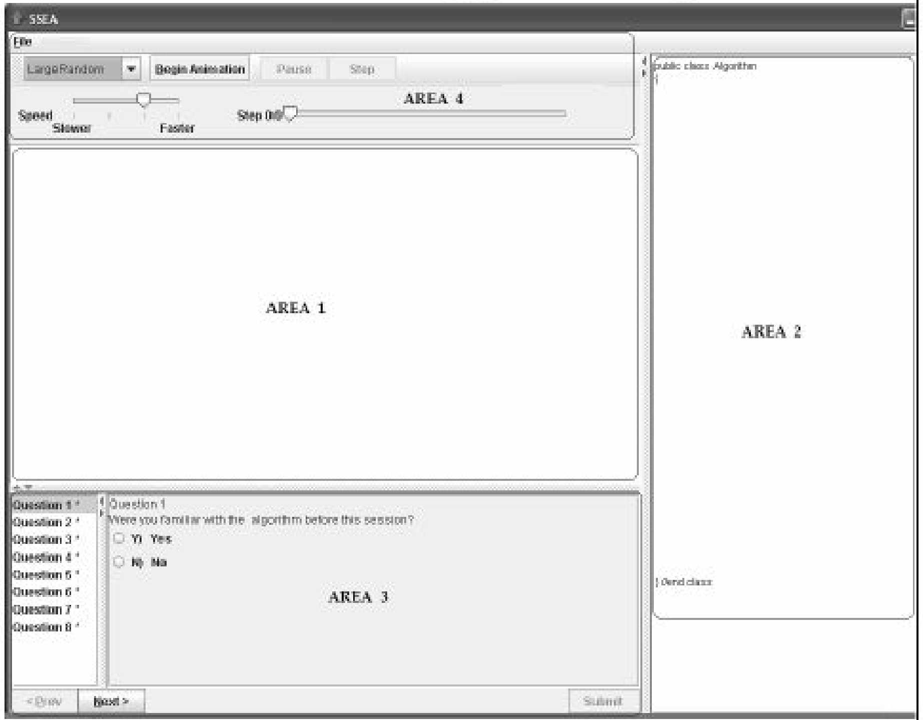


Figure 1. Areas of SSEA program.

Areas in SSEA (see Figure 1)

1. ***Animation Area***: portion where the animation will be played.
2. ***Code Area***: portion where the code for the algorithm is displayed.
3. ***Question Area***: place to answer questions.
4. ***Animation Control Area***: controls to change and interact with animation.

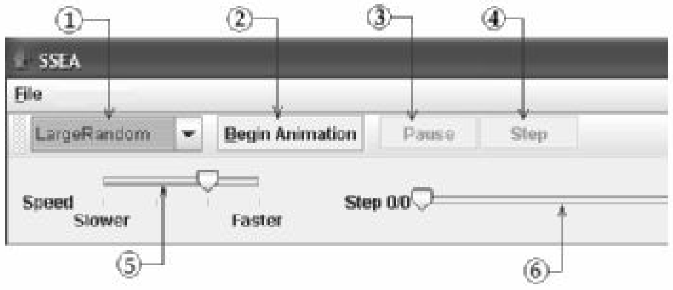


Figure 2. Animation controls.

Animation controls description (see Figure 2)

1. ***Change input sets***: select another item from drop down.
2. ***Begin or End animation.***
3. ***Pause/resume (play)***
4. ***Step:*** step through animation. Must first pause.
5. ***Adjust the speed*** of animation.
6. ***Go to step.*** Move slider back to a previous step.

SCRATCH PAPER

(If you need additional sheets, please ask a researcher).

FEEDBACK FORM

1. What feature(s) did you like best about the algorithm animation? Why?

2. What feature(s) did you like least about the algorithm animation? Why?

3. Did you find any questions, graphics, or other components to be useful? If so, why?

4. Did you find any questions, graphics, or other components to be confusing? If so, why? How would you change it so it is less confusing?

5. Please give us any additional comments about the demo or quicksort animations. What would you definitely change? What would you definitely keep the same?

6. If you were required to answer questions during the animation, please provide feedback about them? Did they help you understand the algorithm better?

7. Please let us know if you have any other comments or questions.

APPENDIX B

CODE FOR VERSION OF QUICKSORT USED IN STUDIES

```
public class Quicksort(){
    public static void main()
    {
        int[] arr;
        quicksort(arr, 0, arr.length - 1);
    }

    private void quicksort(int[] array, int begin, int end)
    {
        if (end > begin)
        {
            // partition the subarray
            int pivotIndex = partition(array, begin, end);

            // now sort the lower partition
            quicksort(array, begin, pivotIndex - 1);

            // now sort the higher partition
            quicksort(array, pivotIndex + 1, end);
        }
    }

    private int partition(int[] array, int begin, int end)
    {
        int firstHigh = begin;
        int pivot = array[end]; // set pivot

        for (int findLow = begin; findLow < end; ++findLow)
        {
            if (array[findLow] <= pivot)// compare
            {
                swap(array, firstHigh, findLow); // swap
                firstHigh++;
            }
        }
        swap(array, firstHigh, pivot);//swap pivot
        return firstHigh;
    }

    private void swap(int[] array, int i, int j)
    {
        int tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
    }
}
```

APPENDIX C

SAMPLE PRE- AND POST-TESTS

Pre-test Questions for Quicksort Algorithm

1) Which best describes how quicksort works?
   a) It partitions an array into two subarrays and uses the median value of the array as the pivot.
   b) For each element,x, in the array, move x to the index equal to the number of elements that are less than x.
   c) It partitions the array into two subarrays and sorts the subarrays independently.
   d) It swaps adjacent items that are out of order.

Answer: c

2) How is the pivot used?
   a) To identify the largest element of the array.
   b) To identify the smallest element of the array.
   c) To identify the median element of the array.
   d) To separate the elements of the array into two subarrays.

Answer: d

3) Given the sequence 7 8 6 2 1 9 4 3. If 3 is chosen as the pivot, which of the following could be the new order after the first call to the partition function?
   a) 2 1 3 7 8 9 4 6
   b) 7 6 2 1 8 4 3 9
   c) 7 8 6 2 3 1 9 4
   d) 1 7 8 6 2 3 9 4

Answer: a

4) When does the worst-case time for quicksort occur for an array of n elements?
   a) When the pivot is always the largest or smallest element in the active partition.
   b) When the input size is a power of 2.
   c) When the partition splits the array into 2 subarrays of equal lengths.
   d) There is no predictor for worst-case time.

Answer: a

5) When does the best-case time for quicksort occur for an array of n elements?
   a) When the pivot is always the largest or smallest element in the active partition.
   b) When the input size is a power of 2.
   c) When the partition splits the array into 2 subarrays of equal lengths.
   d) There is no predictor for best-case time.

Answer: c

6) The quicksort algorithm can best be described as:
   a) selective
   b) recursive
   c) iterative
   d) abstract

Answer: b

7) During a run of the partition function each number is compared to:
   a) its neighbor
   b) all other numbers
   c) itself
   d) the pivot

Answer: d

8) The outcome of partitioning is:
   a) to place all numbers in sorted order
   b) that no number in the lower partition is larger than any number in the higher partition
   c) to place half of the numbers into the left partition
   d) to place all numbers larger than pivot in sorted order

Answer: b

Post-test Questions for Quicksort Algorithm

****Please answer the following questions based on the version of the quicksort algorithm depicted in the animation.****

1) Which best describes the correct order of events of the version of quicksort you just viewed.
   - I. Call quicksort on the higher partition.
   - II. Compare elements to the pivot. If it is less than or equal in value, then swap element into the lower section of the partition.
   - III. Select a pivot.
   - IV. Call quicksort on the lower partition.
   - V. Swap the pivot into the position between the lower and higher partitions.

   a) I, II, III, IV, V
   b) III, II, V, IV, I
   c) III, II, I, IV, V
   d) II, III, IV, I, V

Answer: b

2) Which element is chosen as the pivot in the active partition?
   a) The leftmost element
   b) A Random element
   c) The middle element
   d) The rightmost element

Answer: d

3) When is the pivot swapped?
   a) When a value less than or equal to the pivot value is found.
   b) When a value greater than the pivot value is found.
   c) At the end of partitioning a subset of the array.
   d) None of the above

Answer: c

4) The pivot is swapped with ___________________.
   a) the first element in the lower partition.
   b) the last element in the lower partition.
   c) the first element in the higher partition.
   d) the last element in the higher partition.

Answer: c

5) The comparison of an element with the pivot is done in which method(s)?
   a) quicksort()
   b) partition()
   c) swap()
   d) a and b

Answer: b

6) What two objects are being compared in the partitioning step?
   a) The element at 'firstHigh' and the element at 'pivot'
   b) The element at 'begin' and the element at 'pivot'
   c) The element at 'firstHigh' and the element at 'findLow'
   d) The element at 'findLow' and the element at 'pivot'

Answer: d

7) Swaps can occur between __________________.
   a) the element at 'firstHigh' and the element at 'findLow'
   b) the element at 'begin'+1 and the element at 'firstHigh'
   c) the element at 'firstHigh'+1 and the element at 'pivot'
   d) the element at 'findLow' and the element at 'firstHigh'+1

Answer: a

8) Assume that the array to be sorted initially contained the following values: 5 7 8 2 9 6. Which of the following will be the higher partition after one invocation of quicksort?
   a) 7 9 8
   b) 6 7 9 8
   c) 7 8 9
   d) 6 7 8 9

Answer: a

9) Given the array 8 3 7 5 1 6 2 4. Which of the following represents the contents of the new array after one invocation of quicksort?
   a) 1 2 3 4 6 8 5 7
   b) 4 3 2 1 5 6 7 8
   c) 4 3 7 5 1 6 2 8
   d) 3 1 2 4 8 6 7 5

Answer: d

10) Which best describes how quicksort works?
   a) It partitions an array into two subarrays and uses the median value of the array as the pivot.
   b) For each element,x, in the array, move x to the index equal to the number of elements that are less than x.
   c) It partitions the array into two subarrays and sorts the subarrays independently.
   d) It swaps adjacent items that are out of order.

Answer: c

11) How is the pivot used?
   a) To identify the largest element of the array.
   b) To identify the smallest element of the array.
   c) To identify the median element of the array.
   d) To separate the elements of the array into two subarrays.

Answer: d

12) When does the worst-case time for quicksort occur for an array of n elements?
   a) When the pivot is always the largest or smallest element in the active partition.
   b) When the input size is a power of 2.
   c) When the partition splits the array into 2 subarrays of equal lengths.
   d) There is no predictor for worst-case time.

Answer: a

13) When does the best-case time for quicksort occur for an array of n elements?
   a) When the pivot is always the largest or smallest element in the active partition.
   b) When the input size is a power of 2.
   c) When the partition splits the array into 2 subarrays of equal lengths.
   d) There is no predictor for best-case time.

Answer: c

14) The quicksort algorithm can best be described as:
   a) selective
   b) recursive
   c) iterative
   d) abstract

Answer: b

15) During a run of the partition function each number is compared to:
   a) its neighbor
   b) the pivot
   c) all other numbers
   d) itself

Answer: b

16) The outcome of partitioning is:
   a) to place all numbers in sorted order
   b) that no number in the lower partition is larger than any number in the higher partition
   c) to place half of the numbers into the left partition
   d) to place all numbers larger than pivot in sorted order

Answer: b

APPENDIX D

SAMPLE POP-UP AND TRADITIONAL QUESTIONS

"Pop-up" - Low Level Questions for Quicksort Algorithm

```
1) What elements were just compared?
   a) 60 and 70
   b) 60 and 40
   c) 70 and 40
   d) 70 and 80

2) What was the last comparison?
   a) Is 70 less than or equal to 20?
   b) Is 40 less than or equal to 70?
   c) Is 40 less than or equal to 20?
   d) Is 20 less than or equal to 40?

3) What variables were compared?
   a) 'firstHigh' and 'findLow'
   b) 'findLow' and 'pivot'
   c) 'firstHigh' and 'pivot'
   d) 'begin' and 'end'

4) What elements were just swapped?
   a) 20 and 10
   b) 70 and 40
   c) 20 and 70
   d) 10 and 40

5) The swap() method was invoked on 70 and __.
   a) 60
   b) 70
   c) 80
   d) It was not invoked.

6) What variables were swapped?
   a) 'firstHigh' and 'findLow'
   b) 'pivot' and 'firstHigh'
   c) 'begin' and 'findLow'
   d) 'findLow' and 'pivot'

7) Which value will be the next pivot?
   a) 80
   b) 50
   c) 70
   d) 60

8) Which of the following variables was pointing to a different object than
   the other three?
   a) 'begin'
   b) 'end'
   c) 'firstHigh'
   d) 'findLow'
```

"Pop-up" - High Level Questions for Quicksort Algorithm

```
1) Why did 70 and 20 just swap?
   a) because the partition() method ended
   b) because 'findLow' > 'firstHigh'
   c) because 'findLow' <= 'firstHigh'
   d) because 'findLow' <= 'pivot'


2) Why did 60 and 80 just swap?
   a) because the partition() method ended
   b) because 'findLow' > 'firstHigh'
   c) because 'findLow' <= 'firstHigh'
   d) because 'findLow' <= 'pivot'


3) The pivot (20) was just swapped, when will
   a) it move again?
   b) at the end of quicksort()
   c) at the end of partition()
   d) never
   e) it cannot be determined


4) All of the numbers to the left of 40
   a) are in the lower partition
   b) are in the higher partition
   c) are greater than the pivot
   d) are sorted


5) Which number will be the pivot next?
   a) 30
   b) 40
   c) 80
   d) 60


6) Why was the quicksort call "skipped"
   a) 'end' <= 'begin'
   b) 'end' > 'begin'
   c) the 'pivot' is out of range
   d) it was not skipped

7) Which subarray of numbers will be sorted next?
   a) the higher partition
   b) the lower partition
   c) the entire array
   d) the faded portion


8) Which variable will be the next to swap with the pivot
   a) 'begin'
   b) 'firstHigh'
   c) 'findLow'
   d)  both a) and c)
```

"Traditional" - Low Level Questions for Quicksort Algorithm

```
1) Which number will be the first pivot for the following sequence:
   2 7 1 4 6 3 5
   a) 2
   b) 4
   c) 5
   d) It cannot be determined because the pivot is randomly selected.
```
Answer: c

```
2) At the beginning of the partition() function, 'begin', 'firstHigh', and
   'findLow' are all set to
   a) the first element in the active partition
   b) the last element of the active partition
   c) the pivot
   d) they are set to different elements
```
Answer: a

```
3) At the end of the partition() function, pivot swaps with which variable?
   a) 'begin'
   b) 'end'
   c) 'firstHigh'
   d) 'findLow'
```
Answer: c

```
4) At the beginning of the partition() function, pivot and which variable
   represent the same object
   a) 'begin'
   b) 'end'
   c) 'firstHigh'
   d) 'findLow'
```
Answer: b

```
5) Other than at the end of partition(), objects are swapped when
   a) findLow is less than or equal to the pivot
   b) findLow is less than or equal to firstHigh
   c) findLow is greater than firstHigh
   d) end is greater than begin
```
Answer: a

```
6) The 'findLow' variable is compared to which variable?
   a) 'begin'
   b) 'end'
   c) 'firstHigh'
   d) 'pivot'
```
Answer: d

```
7) What is the comparison between 'findLow' and the other variable, x?
   a) Is 'findLow' less than x?
   b) Is 'findLow' less than or equal to x?
   c) Is 'findLow' greater than x?
   d) Is 'findLow' greater than or equal to x?
```
Answer: b

8) At step 62 of the 'LargeRandom' data set, what items are swapped?
   a) 20 and 10
   b) 70 and 40
   c) 20 and 70
   d) 10 and 40

Answer: c

"*Traditional*" - *High Level Questions for Quicksort Algorithm*

```
1.   What is the purpose of 'findLow'?
     a. to find the lowest value in the array
     b. to find values lower than the pivot
     c. to find the lowest value in the higher partition
     d. a and c
```
Answer: b

```
2.   The variable 'firstHigh'
     a. points to the first object in the higher partition
     b. points to the last object in the lower partition
     c. swaps with the pivot
     d. a and c
```
Answer: d

```
3.   When might an object appear to swap with itself?
     a. when 'findLow' = 'firstHigh' and 'findLow' <= 'pivot'
     b. when 'findLow' <= 'firstHigh' and 'findLow' <= 'pivot'
     c. when 'findLow' = 'firstHigh' and 'findLow' >= 'pivot'
     d. when 'findLow' >= 'firstHigh' and 'findLow' >= 'pivot'
```
Answer: a

```
4.   The entire quicksort() function is only carried out when
     a.  'begin' is less than 'end'
     b.  'begin' equals 'end'
     c.  'begin' is greater than 'end'
     d. everytime
```
Answer: a

```
5.   The pivot swaps
     a. with 'firstHigh'
     b. at the end of partition()
     c. neither a nor b
     d. both a and b
```
Answer: d

```
6.   When does 'findLow' swap with 'firstHigh'?
     a. when 'findLow' is greater than the pivot
     b. when 'findLow' is less than or equal to the pivot
     c. when 'findLow' is greater than 'firstHigh'
     d. at the end of the partition() method
```
Answer: b

7. The first call to quicksort operates over the entire array. Which subarray will be operated on next?
   a. higher partition
   b. lower partition
   c. all numbers again
   d. it cannot be determined

Answer: b

8. Once the pivot is swapped, when will it move again?
   a. at the end of quicksort()
   b. at the end of partition()
   c. never
   d. it cannot be determined

Answer: c

APPENDIX E

QUICK GLANCE OF ALGORITHM ANIMATION STUDIES BY FEATURE

Feature	Work Done by	Sample results
Animation Display		
Code View	1. Saraiya et al., 2004 2. Naps et al., 2002	1. Code and guide versus data example "covering important cases might provide better conceptual understanding" *(this is based on analysis of 1 conceptual question).* Found code and guide doubled time, but **no** increase in learning. 2. Suggestion (multiple views).
Labels - on objects	1. Badre et al., 1992 2. Lawrence, 1993	1. Should display labels, based on exploratory study. 2. Student preference, but **no** performance difference.
Labels – textual description	1. Rößling et al., 2002 2. Naps et al., 2002 3. Badre et al., 1992 4. Lawrence, 1993	1. Suggestion 2. Suggestion 3. Should display labels, based on exploratory study. 4. Textual and visual cues **significantly** outperformed unlabeled animations
Auralization	1. Brown et al., 1991	1. Used sound to reinforce or replace graphics and signal conditions. Informal evaluation - "sound will be a powerful technique for communicating information."
Graphical Object	1. Lawrence, 1993	1. Dots versus sticks made **no** performance difference; students preferred sticks.
Object orientation	1. Lawrence, 1993	1. Vertical versus horizontal made **no** difference; students preferred vertical.
Data Set Size	1. Lawrence, 1993	1. **No** difference.
Color	1. Brown et al., 1991 2. Lawrence, 1993	1. Used color to show state, unite multiple views, highlight, emphasize patterns, show history; no empirical study 2. Viewers of monochrome display **significantly** outperformed viewers of color displays.
Visual Cues	1. Lawrence, 1993	1. Textual and visual cues **significantly** outperformed unlabeled animations.

Feature	Work Done by	Sample results
Structural view of algorithm.	1. Rößling et al., 2002	1. Suggestion (should allow user to jump to any point of execution by clicking on the display).
Algorithm Performance	1. Naps et al., 2002	1. Suggestion
Execution History	1. Naps et al., 2002	1. Suggestion
Interaction		
No Viewing		
	1. Byrne et al., 1999 2. Grissom et al., 2003 3. Fleischer et al., 2004 4. Hansen et al., 2002 (3 studies) 5. Hansen et al., 2002	1. Prediction plus animation **significantly** outperformed other three groups; 2x2 animation versus text/static and passive versus prediction. 2. No viewing versus Viewing versus Responding; found greater improvement between pre- and post-test for increasing levels of engagement. Held at 3 universities with 4 different professors. No viewing group had much higher pre-test scores. 3. No viewing versus AA that actively engages; **no** difference. 4. **Significant** difference in 2 of 3 text versus full version of HalVis (animation, text, static images, audio narratives, interactive questions). The 3rd in which text was "carefully designed" showed **no** difference in learning. 5. **Significant** difference for HalVis over lecture

Feature	Work Done by	Sample results
Viewing		
Control Direction	1. Saraiya et al., 2004 2. Rößling et al., 2002 3. Naps et al., 2002	1. Providing step back feature **did not** improve performance. 2. Suggestion (rewind). 3. Suggestion (execution control).
Control Speed	1. Saraiya et al., 2004 2. Rößling et al., 2002 3. Naps et al., 2002	1. Providing “absolute control *(stepping)* on the pace of the AV proved to make a **significant** difference”. 2. Suggestion (view animation in smooth motion or discrete steps). 3. Suggestion (execution control).
Other	1. Lawrence et al., 1994 2. Byrne et al., 1999 3. Grissom et al., 2003 4. Fleischer et al., 2004 5. Hansen et al., 2002	1. Lecture plus slide group **slightly** outperformed lecture plus animation group. 2. Prediction plus animation **outperformed** other three groups; 2x2 animation versus text/static and passive versus prediction. 3. No viewing versus Viewing versus Responding found **greater improvement** between pre- and post-test for increasing levels of engagement. Held at 3 universities with 4 different professors. No viewing group had much higher pre-test scores. 4. No viewing versus AA that actively engages users; **no** difference. 5. **Significant** difference for HalVis versus XTango + handout.

Feature	Work Done by	Sample results
Responding		
Questions – Pop-up (Prediction, Responsive, "What-if", High-level, Low-level, Feedback)	1. Rößling et al., 2002 2. Naps et al., 2002 3. Byrne et al., 1999 4. Naps et al., 2000 5. Jarc et al., 2000 6. Grissom et al., 2003	1. Suggestion – "stop-and-think" to answer next step of algorithm. 2. Suggestion – dynamic, "pop quiz" questions; provide dynamic feedback. 3. Prediction plus animation **significantly** outperformed other three groups; 2x2 animation versus lecture and passive versus prediction. 4. No control group; 5 participants had "stop-and-think" questions and performed moderately. 5. Predictive group performed **slightly worse** than passive animation group 6. No viewing versus Viewing versus Responding found **greater improvement** between pre- and post-test for increasing levels of engagement. Held at 3 universities with 4 different professors. No viewing group had much higher pre-test scores.
Changing		
Enter input set	1. Saraiya et al., 2004 2. Rößling et al., 2002 3. Naps et al., 2002 4. Lawrence, 1993	1. Participants who were given an example performed **slightly better** than those who gave their own input (no combination of both given and user input); **significant improvement** on procedural questions. 2. Suggestion 3. Suggestion 4. Viewers who entered their own input set had **significant** improvement over passive viewers.

Feature	Work Done by	Sample results
Enter input set (cont.)	5. Lawrence et al., 1994	5. *Lecture plus lab – active* group **outperformed** *lecture plus slide* (screen shots from animation) and *lecture plus lab – passive*.
Constructing		
Direct Generation (map program to visualization)	1. Naps et al., 2002 2. Stasko, 1996 3. Hundhausen et al., 2000	1. Suggestion (students should build own visualizations). 2. Informal evaluation; students became competitive, had almost perfect scores on related questions on final, enjoyed class. 3. **No** significant difference for AVs constructed in 2.5 hours.
Hand Construction (art supplies or animation editor)	1. Naps et al., 2002 2. Hubscher-Young et al., 2003	1. Suggestion (students should build own visualizations). 2. **Significant** improvement for students who created, shared, and evaluated an AV.
Presenting		
	1. Hubscher-Young et al., 2003	1. **Significant** improvement for students who created, shared, and evaluated an AV.
Usability / Other		
Ease of use		
Expected use	1. Stern et al., 2005	1. Participants **did not** use the system, AIA, as expected.
Intended Level of User	1. Naps et al., 2002	1. Suggestion (Intended user level should be indicated).
Window Management		
Make widely available (i.e. use Java for platform independency or applets for online availability)	1. Rößling et al., 2002	1. Suggestion.
General purpose (show many different types of visualizations, but user only has to learn system once)	1. Rößling et al., 2002	1. Suggestion.
Additional Handout	1. Lawrence, 1993	1. Animation vs. Handout first made **no** difference.