# Integrating Algorithm Visualization Technology into an Undergraduate Algorithms Course: Ethnographic Studies of a Social Constructivist Approach

CHRISTOPHER D. HUNDHAUSEN

Laboratory for Interactive Learning Technologies
Information and Computer Sciences Department
University of Hawai'i
1680 East West Road, POST 303D
Honolulu, HI 96822
(808) 956-3887
(808) 956-3548 (Fax)
hundhaus@hawaii.edu

## Abstract

Algorithm visualization (AV) software graphically illustrates how algorithms work. Traditionally, computer science instructors have used the software as a visual aid in lectures, or as the basis for interactive laboratories. An alternative approach, inspired by Social Constructivist learning theory, is to have students construct and present their own visualizations. Notice that, in this alternative approach, rather than acting as a *knowledge conveyer* that transfers an expert's mental model of an algorithm to students, AV software grants students access to forms of expert course participation typically reserved only for instructors. To explore this alternative approach, I conducted a pair of ethnographic field studies in a junior-level algorithms course that included AV construction and presentation assignments. Through a broad range of field techniques, including participant observation, interviewing, videotaping, and diary collecting, I gained crucial insights into three key research questions surrounding AV construction and presentation assignments: (1) Do the assignments promote activities that are relevant to the community being reproduced through an undergraduate algorithms course?; (2) Do the assignments promote *learning*—that is, do they help students to participate more fully in that community?; and (3) How should AV technology be designed so as to support AV construction and presentation? My most significant finding was that, when used within the context of AV construction and presentation exercises, conventional AV

software can actually *distract* students from concentrating on activities and concepts relevant to an undergraduate algorithms course. When supported by an alternative, "low tech" version of AV technology, however, AV construction and presentation appear not only to focus students on relevant activities and concepts, but also to enable them to participate more extensively the course, thus contributing to their learning.

**Keywords**: simulations; cooperative/collaborative learning; evaluation of CAL systems; teaching/learning strategies; pedagogical issues

# 1. Introduction

Over the past two decades, a legacy of algorithm visualization (AV) software has been developed for computer science education (see, e.g., Brown, 1988; Naps, 1990; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1990). The purpose of this software has been to provide an environment for constructing and interactively exploring graphical representations of algorithms in action. Traditionally, computer science instructors have used the software to construct visualizations that are later used either as visual aids in lectures, e.g., (Brown & Sedgewick, 1984), or as the basis for interactive labs, e.g., (Naps, 1990).

Despite the enthusiasm and high expectations of AV software developers, a review of 22 experimental evaluations (Byrne, Catrambone, & Stasko, 1999, Study I and II; Crosby & Stelovsky, 1995; Gurka, 1996; Hansen, Narayanan, & Schrimpsher, 2000, Study I – VIII; Hundhausen & Douglas, 2000b; Jarc, Feldman, & Heller, 2000; Kann, Lindeman, & Heller, 1997; Lawrence, 1993, Ch. 4–9; Stasko, Badre, & Lewis, 1993) casts doubt on the software's pedagogical benefits. Indeed, only 13 of those experiments (Byrne et al., 1999, Study I and II; Crosby & Stelovsky, 1995; Hansen et al., 2000, Study I, II, IV, V, VII, VIII; Kann et al., 1997; Lawrence, 1993, Ch. 6, 7, 9) showed that some aspect of AV technology or its pedagogical application significantly impacted learning outcomes.

Further analysis of these experiments suggests that they fall into two broad categories based on the factors they identify as critical to the experimental evaluation of AV. Lawrence (1993, Ch. 4, 5, 7,

8), Stasko, Badre, and Lewis (1993), Gurka (1996), and Hansen et al. (2000, Study VI – VIII) varied *representational characteristics* of the learning materials, including (a) text versus animation, (b) text-first or animation-first, and (c) various graphical attributes of animations. By contrast, a second group of studies (Byrne et al., 1999, Study I and II; Hansen et al., 2000, Study I – V; Jarc et al., 2000; Kann et al., 1997; Lawrence, 1993, Ch. 6 and 9) varied level of *learner involvement*. In addition to having learners passively view an animation, these studies involved learners more actively by having them (a) construct their own data sets, (b) answer strategically chosen questions about the algorithm; (c) make predictions regarding future algorithm behavior, or (d) program the algorithm.

If one considers the results of the experiments vis-à-vis these two categories (Figure 1), a notable trend emerges: experiments that manipulate learners' level of involvement have consistently yielded significant results, whereas experiments that have manipulated representation have not. This suggests that what learners do, not what they see, may have the greatest impact on learning—an observation that well accords with *constructivist* learning theory (see, e.g., Papert, 1980).

Given this trend, computer science instructors have good reason to consider pedagogical approaches that get students maximally involved in the process of visualizing an algorithm. To that end, Stasko (1997) advocates "visualization assignments" in which students to construct their own visualizations of the algorithms under study. Inspired by an alternative, social version of constructivist learning theory (Lave & Wenger, 1991), I have become interested in a teaching approach that takes Stasko's visualization assignments one step further by having students not only construct their own visualizations, but also *present* their visualizations to their instructor and peers for feedback and discussion. According to Social Constructivism, AV software is pedagogically valuable in such assignments insofar as it enables students to participate in a course in ways that increasingly resemble the ways in which teachers participate. In facilitating such participation, AV technology serves to mediate meaningful learner-learner and learner-expert conversations about algorithms.

Clearly, adopting Social Constructivism as one's theoretical orientation requires a corresponding shift in one's method for evaluating effectiveness. Indeed, rather than evaluating traditional

learning outcomes through controlled experimentation, as have the legacy of studies reviewed above, one requires a method that examines the role of AV technology both within the broader context of the community in which the technology facilitates participation (a computer science course), and with respect to its role in mediating conversations about algorithms. In other words, one needs to conduct an *ethnographic field study* of a computer science course that includes AV construction and presentation activities.

In this article, I present a pair of ethnographic field studies that explored two separate offerings of a junior-level algorithms course that included AV construction and presentation assignments.[1] The two courses were taught by the same instructor during consecutive ten-week quarters at the University of Oregon. During the first quarter of the study, to which I shall refer as Study I, the instructor and I modeled the animation assignments largely after Stasko's (1997) recommendations. However, our experiences during that quarter led us to revise significantly the animation assignment format and requirements for the subsequent academic quarter, to which I shall refer as Study II

With respect to the *effectiveness* of such assignments, two key research questions guided my fieldwork:

1. Do AV construction and presentation assignments promote activities that are *relevant* to the community being reproduced through an undergraduate algorithms course?

2. Do AV construction and presentation assignments promote *learning*—that is, do they help students to participate more fully in that community?

In addition, I was interested in the design of effective AV technology to support this approach. This raised a third research question:

3. How should AV technology be designed so as to support relevant AV construction activities, and so as to mediate pedagogically effective conversations about algorithms?

---

[1]For a fuller treatment of these studies, see (Hundhausen, 1999, ch. 4 and app. A–B).

As I shall illustrate, observations made during my fieldwork provide crucial insights into these research questions. The most significant of these is that, when used within the context of AV construction and presentation exercises, conventional AV technology can actually *distract* students from concentrating on activities and concepts relevant to an undergraduate algorithms course. When supported by an alternative, "low tech" version of AV technology, however, AV construction and presentation appear not only to focus students on relevant activities and concepts, but also to enable them to participate more extensively the course, thus contributing to their learning, in the Social Constructivist sense.

This article is organized as follows. I begin, in Section 2, with some essential background on the studies: what was covered in the algorithms course and how the course was organized; what AV technology was used and how it was used; and who the informants in the study were. Section 3 considers what made these studies "ethnographic" by describing the various ethnographic field techniques I employed. Section 4 presents my key observations, which relate to students' activities and experiences in the process of constructing and presenting their AVs. Section 5 considers these observations vis-à-vis the three key research questions posed above. Section 6 reviews related work. Finally, Section 7 presents key conclusions and highlights promising directions for future research.

## 2. Background

CIS 315, "Algorithms," typifies the third year undergraduate algorithms course taught within the computer science departments of American universities. In the course, students explore efficient algorithmic problem-solving techniques, including divide-and-conquer, dynamic programming, and greedy approaches. The course emphasizes that such techniques are generally applicable to wide classes of problems; the trick is to recognize a problem as being a candidate for a certain technique, and then to apply the appropriate technique to solve the problem. The course also stresses the importance of the formal reasoning skills necessary to talk precisely about the correctness and efficiency of the algorithms under study. Formal proofs of correctness, and precise statements about efficiency (using Big-O notation), are thus important components of the course.

The CIS 315 course in which I conducted my fieldwork adopted a standard text (Cormen, Leiserson, & Rivest, 1990), and revolved around three fifty-minute lectures per week. An additional 50-minute discussion period provided an opportunity for students, the teaching assistant, and occasionally the instructor to come together to discuss problems of current interest. Grading was based on regular problem sets, a midterm, a final exam, and various algorithm visualization assignments.

Below, I briefly describe the students and instructor who participated in the two courses I observed, and the animation assignments that provided a backdrop for my fieldwork.

## 2.1 Informants: Students and the Instructor

Thirty-eight computer science majors were enrolled in the first CIS 315 class I observed; forty-nine were enrolled in the second. Prior to their enrollment in the course, these students were required to complete both a math sequence that culminated in a standard discrete mathematics course, and a computer science sequence that culminated in a standard 300-level data structures course. Students ranged in age from 20 to over 40, with most of them closer to 20. Most students in the courses were male; five females were enrolled in each of the two courses.

John Lane,[2] the course instructor, was a tenured professor who had been teaching the algorithms course at the university for over 12 years. In addition to holding regular office hours, John gave nearly all of the course lectures, did some of the grading, and led in some of the weekly discussion sections. Tom, a graduate student in the department, was the teaching assistant for the course both terms. In addition to holding weekly office hours, Tom did most of the grading, led most of the weekly discussion sections, and occasionally gave lectures when John was out of town.

## 2.2 Algorithm Visualization Assignments and Supporting Technology

Recall that, according to Social Constructivism, AV technology is effective insofar as it provides students with access to fuller course participation—that is, insofar as it enables students to participate in the course in ways that increasingly resemble the ways in which an instructor

participates. In an algorithms course that makes use of AV technology, an instructor typically creates visualizations for classroom use, and then uses them as a visual aid in lectures. Thus, on the Social Constructivist view, AV technology holds promise as a learning tool through its ability to facilitate student access to those two expert activities: AV construction and presentation.

Following this Social Constructivist recommendation, Professor Lane and I devised AV assignments that included both a construction component, and a presentation/discussion component. However, these components differed substantially in each of the course offerings that I studied.

### 2.2.1 Study I Visualization Assignments

During the first term of my fieldwork, the construction component of the assignments was based on Stasko's (1997) recommendations. In particular, students had to complete three different animation assignments. The first assignment was intended to familiarize them with the animation construction environment (discussed below) that they would be using for subsequent assignments. For the second and third assignments, students, who were allowed to work in groups, were asked to choose an algorithm that made use of one of the problem-solving techniques (divide-and-conquer, greedy, dynamic programming) covered in class. They were then asked to create an animation for the algorithm they had chosen, keeping in mind that the animation should work for general input. Departing from Stasko's recommendations, we also required students to present the animation to their instructor classmates at one of the animation presentation sessions to be scheduled near the end of the term.

Students used the *Samba* (Stasko, 1997) animation package to construct and present their visualizations. A front-end interpreter to the Polka animation system (Stasko & Kraemer, 1993), Samba supports two-and-a-half dimension, multiple-window, color animations. The Samba interpreter takes as input a text file containing a series of Samba commands (one per line), and generates a corresponding animation. Figure 2 presents a fragment from a sample Samba script, along with a snapshot of the animation generated by the script.

---

[2]In the interest of preserving their anonymity, I use pseudonyms to refer to all of my informants.

Once a Samba animation has been programmed, it can be viewed using the tape recorder-style interface illustrated in Figure 3. The interface allows one to start, pause, and step through the animation (one frame at a time), and to adjust the execution speed. An additional set of controls in each animation window [see bottom of window presented in Figure 2(b)] allows one to zoom in and out of the view, and to pan around within the view.

#### 2.2.2 Study II Visualization Assignments

For the subsequent offering of the course, Professor Lane and I opted to revise the animation assignments in three significant ways. First, we decided to drop two of the three animation assignments for the subsequent term; only a final animation assignment was retained. Second, we dropped the requirement for input generality; students were instead asked to choose carefully a few examples to animate, and then to focus on animating those examples well.

Third, we turned the animation assignment into a two-phase project. For the first, "animation prospectus" phase of the project, student groups were asked to use low-tech materials (e.g., transparencies, pens, construction paper, scissors) to construct "visualization storyboards" (Douglas, Hundhausen, & McKeown, 1995) of their proposed animations. They were asked to present these storyboards to Professor Lane, me, and small groups of interested students during storyboard presentation sessions scheduled at roughly the halfway point of the term. For the second, "Samba animation" phase of the project, students were asked to implement their storyboards as Samba animations, taking into consideration the suggestions and feedback they had received from their storyboard presentations.

## 3. Field techniques

In my fieldwork, I played the dual-role of *student observer* and (volunteer) *teaching assistant for algorithm animation*. As a student observer, I sat in on lectures and took notes; interacted with students before and after lectures, and occasionally when I ran into them in the computer science department; and arranged to observe and work with certain groups of students as they worked on animation assignments in the undergraduate computer lab.

As the teaching assistant for algorithm animation, I collaborated with the instructor in the development of the algorithm animation curriculum; set up and maintained the Samba software used for the algorithm animation assignments; gave introductory lectures on algorithm animation and the course animation assignments; made myself available via e-mail, and before and after class, for questions regarding algorithm animation; and interacted regularly with the instructor regarding a variety of issues surrounding the animation assignments.

In this dual-role, I made use of seven different ethnographic field techniques. First, I made extensive use of *participant observation* to participate in and observe the algorithm animation-related activities of both students and the instructor. Second, I used two different kinds of interviewing techniques to elicit my informants' perceptions and experiences. In my day-to-day interaction with students and the instructor, I tended to ask a lot of questions on an informal basis (*informal interviewing*). On several occasions, I followed up on the important themes and issues that emerged from those informal inquiries by audiotaping (and subsequently transcribing and analyzing) *semi-structured interviews* with students and the instructor. Third, during Study I, I administered two brief on-line *questionnaires* to the members of a volunteer mailing list. These questionnaires elicited students' general impressions regarding the algorithm animation assignment, what activities they performed, and estimates of the amount of time spent on each activity.

Fourth, I took extensive *fieldnotes* during both terms of the fieldwork, both during lectures, and after my discussions with Professor Lane and my student informants. Fifth, I *audio-* or *videotaped* (and subsequently transcribed and analyzed) all of the storyboard and final animation presentation sessions that were held during both terms of the fieldwork. Sixth, I *collected and analyzed artifacts*—both the executable animations that students handed in during Study I and Study II, and the low-tech storyboards that students presented during Study II. Finally, in Study II, I *collected and analyzed diaries* that students were required to hand in as part of the assignment. Their diaries documented what they did for the animation assignment, what problems they encountered, and how much time they spent.

# 4. Observations

In the two courses I observed, a total of 83 student groups (one to four students each) constructed animations of 22 algorithm themes. Figure 4 graphs the number of student groups that animated each algorithm theme. As the figure indicates, the QuickSelect algorithm, Dijkstra's algorithm, Kruskal's and Prim's minimum spanning tree algorithm, and breadth-first and depth-first search all proved to be particularly popular. Notice also that the algorithm themes that students chose were representative of the major problem-solving techniques studied in the course: divide-and-conquer, greedy, dynamic programming, and graph algorithms.

Figure 5 presents the 83 student animation projects according to the representations they used to portray the target algorithms. Most of the animation projects, as the figure indicates, employed *geometric representions*; they depicted their target algorithms using the canonical, purely geometric representations that appear in the course textbook.

For example, many of student groups animated Kruskal's and Prim's minimum spanning tree algorithms by (a) representing an input graph as labeled circles connected by lines, and (b) shading edges and nodes in some way to illustrate the minimum spanning tree as it was being constructed (see Cormen et al., 1990, pp. 506–508).

In contrast, a relatively small number of student groups opted to portray their algorithm in terms of a *story*, in which real or fictitious human beings were engaged in some problem-solving venture. Derived either from the real world or from a fantasy, these story-based animations had at least one of the following two properties, which distinguished them from their canonical geometric counterparts. First, their storyline served to motivate the use of the target algorithm by providing a rich backdrop for its application. Second, their internal logic and structure paralleled the target algorithm's internal logic and structure; the storyline was an analogy for the algorithm. An example

of a story-based animation with both of these properties is the story of Knuth's Ark, which portrays the algorithm for solving the longest common subsequence problem[3]:

> The world floods again, and Donald Knuth[4] builds an ark. Loaded with a pen of animals, Knuth's Ark sails from island to island in search of surviving animals. Like Noah, Knuth's goal is to save pairs of like animals, so that they might breed and eventually replenish the population. One day, Knuth's Ark lands on an island, and Knuth sights a herd of wild animals—lions, tigers, giraffes, among others. Given the animals already aboard Knuth's Ark, which animals on the island should Knuth select so that he has the most pairs of like animals?

Recall that, in the courses I observed, the animation assignments actually had two distinct components: animation construction and animation presentation/discussion. In the remainder of this section, I first present my observations of students' animation construction activities. I then turn to my observations of the animation presentation and discussion sessions.

## 4.1 Animation Construction

Based on the diary data I collected in Study II, Figure 6[5] compares the average amount of time students spent implementing their final animations in Samba, and the average amount of time students spent constructing their storyboards with art supplies. As the figure illustrates, the average amount of time students spent on those two construction activities varied substantially.[6] In light of the large difference between the average amount of time students spent constructing storyboards, and the average amount of time students spent implementing Samba animations, the obvious question to ask is, "How, exactly, did students spend their time in each of those activities?" Below, I elaborate further on the activities in which students engaged as they and as they constructed storyboards out of art supplies, and as they implemented animations in Samba.

---

[3]Given two sequences of objects, what is the longest (not necessarily contiguous) subsequence that the two sequences have in common? The problem can be solved efficiently using dynamic programming; see, e.g., (Cormen et al., 1990, pp. 314–319).

[4]Knuth is a famous algorithmatician and pioneer of the contemporary approach to algorithm analysis; see his multi-volume set *The Art of Computer Programming* (Knuth, 1973).

[5]Twenty-seven of the 47 students registered in the course turned in diaries documenting their storyboard-construction activities (7), their Samba implementation activities (7), or both (13). Thus, the data reported in this figure reflects a sample size of 20—a significant portion of the total population.

[6]These numbers serve as conservative estimates of the time students spent on the animation assignments of Study I. Indeed, given the more stringent requirements of the Study I assignments (*viz.*, that the Study I assignments required all animations to work for general input, whereas that the Study II assignments required students to construct animations that worked for only a few input data sets), and given the fact that students were able to flesh out many of the details of their Samba animations during the storyboard phase of the Study II assignment, one would actually expect students to have spent *more* time on the Study I assignments than they spent on the Samba implementation component of the Study II assignment.

### 4.1.1 Storyboard Construction Activities

Based on an analysis of student diaries, Figure 7 presents a taxonomy of the activities in which students engaged in the storyboard phase of the Study II animation assignment. As suggested by the taxonomy, students' storyboard construction activities entailed researching, talking about, and preparing their presentations on algorithms. To prepare their presentations, most student groups (22 of 24) prepared visual aids in advance. Nearly half of these groups (11) used black-and-white transparencies generated by some sort of graphics editor or drawing program. The others made use of a variety of storyboard materials, ranging from hand-drawn transparencies (4 groups), to hand-drawn or computer-generated sheets of paper (4 groups), to large poster board with hand-drawn illustrations (2 groups).

As the following section makes clear, storyboard preparation activities differed markedly from the activities in which students engaged as they implemented Samba animations.

### 4.1.2 Samba Construction Activities

Derived from interview and questionnaire data collected in Study I, Figure 8 presents a taxonomy of the activities in which students engaged as they implemented input-general animations in Samba. As the figure indicates, three high-level activities were involved. First, since the assignment statements that Professor Lane and I gave to students were open-ended and vague, students had to decide on a specific project. While some students relished the flexibility the projects afforded, others were stymied by it. As one student confided in a questionnaire response, "Just deciding on which were the important parts [of the assignment] to get done was a problem. Lots of people didn't quite know what [Professor Lane] wanted to be turned in."

Recall that the Study I assignments required students' animations to work for general input. To facilitate input generality, students engaged in the second high-level activity depicted in Figure 8: implementing a "driver" algorithm.[7] Unfortunately, Professor Lane did not supply students with

[7]The ultimate purpose of such a driver algorithm was to produce automatically a Samba animation for any input. To accomplish this, students had to annotate the driver algorithm with statements that print out Samba code at interesting points. This technique, which I labeled *direct generation* in Chapter II, was originally pioneered by Brown (1988).

pre-programmed algorithms to animate. While some students were able to "borrow" source code for their algorithms from various sources (e.g., friends, the World Wide Web), others dedicated substantial amounts of time to implementing their algorithms, which also entailed the secondary activities of writing input routines (to accept input data), and debugging the algorithm. In fact, according to my questionnaire data, algorithm programming activities, on average, accounted for the largest portion of the overall time spent on the second Study I assignment.

The third and final high-level animation activity was to implement the animation. Animation programming involved four main activities. First, students had to lay out their animations on the screen. To facilitate the placement of objects on the screen, the Samba language supports a Cartesian, real-numbered coordinate system. Since they did not always find this coordinate system to be easy to work with, students often noted that they needed much trial-and-error in order to get their animation layouts to look right.

Next, students had to figure out how to get their animation to work for general input. This entailed writing general-purpose graphics routines. Such routines had to be parameterized, so that they could lay out and update the animation for any reasonable input. Following sound principles of structured programming, students sometimes designed a suite of graphics classes for this purpose. One student, in fact, reported that he used an object-oriented design tool to create a 12,000 line library of parameterized graphics classes for Samba; he subsequently made the library available to the class.

Third, students spent time debugging their animations. In the case of the Samba animation assignments, debugging took on a new twist, since observed problems in students' animations could have one of two causes: (a) a bug in the underlying algorithm, or (b) a bug in the mapping of the algorithm to the animation. While my fieldwork did not include a detailed investigation of the ways in which students went about the debugging task, interview data indicate that students actually used their animations as a resource for debugging their algorithms. Consider, for example, the following description one of my informants offered of her group's debugging process:

> [I]n the beginning, . . .you're making sure you just have the animation right. But after you're pretty sure you've got the animation right, you can use that to debug the more important

> details of your code. Like, once we. . .figured out how to delete [the edges in our graph] from the screen, we had to make sure we were actually deleting them in [the algorithm] as well. If it didn't disappear from the screen, it was probably because we didn't delete it from the algorithm.

Finally, in addition to programming and debugging their animations, students spent time "tweaking"—that is, fine-tuning their animations so that they met their personal presentation standards. I learned, in fact, that some students actually enjoyed the process. As one of my informants wrote after completing the third Study I animation assignment, "I spent the same amount of time [as I did on the previous assignment] getting the geometry to line up so the whole presentation was clean, a process I enjoy a lot. . .If you gotta present something, you want it to look nice, to look clean." In addition to being potentially fun, the tweaking process proved time consuming. This was because of the lengthy *compile-run-run* cycle required to test a modification to a Samba animation (the first run generates the Samba trace; the second run allows one to view the animation in Samba). As one student stated in a questionnaire response,

> I didn't like. . .the amount of time it took to set up the actual C++ algorithm, and how long it took to compile and run the program just to check for a change in one tiny detail. One detail can make or break the Samba code (i.e., a detail like changing the current view can cause hundreds of lines of code to be ignored), and this detail is sometimes hard to spot when debugging.

Another student had a similar experience, confessing that he spent so much time tweaking his second animation assignment that he was "too embarrassed to say" just how much time he actually spent.

In Study II, the absence of an input generality requirement meant that students had greater freedom in their choice of animation implementation strategies. An analysis of the Study II animations that students handed in indicates that students took five alternative approaches to the implementation of their final animations.

The most popular approach, taken by 11 student groups, was to write a C++ "driver program" that produced a Samba trace file illustrating the algorithm for a single set of input data. A second strategy, adopted by six student groups, was to implement a canned animation in Java, thus avoiding Samba altogether. Third, although input generality was not a requirement, four student groups nonetheless opted to implement input-general animations using the same strategy used by

students in Study I. Fourth, and in stark contrast to the students who wrote general-purpose animations, four student groups avoided the need to implement any C++ code at all by hand-coding their animations directly in the Samba scripting language. Finally, one group wrote a custom animation layout tool as a front-end to Samba. Using this tool, they were able to specify their animation using a graphics editor coupled with a custom command language.

While some of these strategies successfully avoided the need to implement a general-purpose driver program, they all required students to engage in many of the same activities described in the taxonomy of Figure 8. In fact, my analysis of student diaries suggests that students spent far and away the most time on the programming activities necessary to get their animations up and running. As was the case in Study I, these programming activities included writing and modifying algorithms, annotating algorithms with Samba statements; debugging their animations; and tweaking their animations.

## 4.2 Animation Presentation and Discussion

The animation presentation/discussion sessions were a much-anticipated capstone of students' animation-building efforts. Having spent potentially significant amounts of time preparing for their presentations, students often looked forward to showcasing what they had done, and to receiving feedback. Based largely on post-hoc review of the videotaped footage, the following two subsections take a closer look at the storyboard and Samba presentation sessions.

### 4.2.1 Storyboard Presentations

In presentation sessions that lasted between ten and twenty minutes, students storyboarded their animations by presenting snapshots of the animation at key points in its execution. While some students created separate illustrations for key frames of the animation, others made use of pens and cut-out figures to update a single illustration. For example, in the storyboard of a greedy job scheduling algorithm, student presenters slid cut-outs of the jobs to be scheduled along a timeline; a cut-out was deposited in its rightful place on the timeline if it could be scheduled, or slid off of the timeline if it could not be scheduled. Similarly, many groups used pens to mark up their

storyboards as they unfolded. For instance, in storyboards of graph algorithms, vertices and edges were often shaded or circled to indicate whether they were visited or chosen.

Students provided verbal play-by-play narration of their storyboards. As storyboard events unfolded, students made extensive use of deictic gesture, using both the tips of pens and their fingers to coordinate their narration and explanations with objects in their storyboards. Likewise, Professor Lane often pointed to objects in the storyboard when he had questions, or when he made suggestions. In addition, since many storyboard drawings were essentially static, students frequently used gestures to impart a degree of dynamism on storyboard objects. For example, if consecutive snapshots portrayed the same object at successive points in the animation, students often made sweeping gestures to indicate how the object got from the first point to the second point.[8]

Student-professor discussions were often lively during the storyboard presentations. In these discussions, three major themes emerged. First, audience members frequently broke in to ask clarifying questions, which served to clarify various aspects of storyboard presentations, including (a) the sequence of storyboard events (e.g., "Is the table updated before or after you draw the arrow?"); (b) what happens between storyboard snapshots (e.g., "How do you get from that slide to this one?); (c) how, precisely, the final animation will unfold (e.g., "How will you actually show that table update?"); and (d) the significance of attributes of storyboard objects (e.g., "Why is that edge colored red?").

Second, John frequently offered suggestions for improving a storyboard's design. Less frequently, students explicitly elicited suggestions regarding the design of their storyboards. Some of these suggestions were one-sided monologues; John did the talking, and students did the listening. In contrast, other suggestions were collaborative achievements; they arose out of discussions in which Professor Lane and student presenters considered alternative designs.

---

[8]Note that these observations concur with those of prior research into storyboard presentations (Chaabouni, 1996; Douglas, Hundhausen, & McKeown, 1996 ).

Analysis of videotaped footage of the storyboard presentations suggests that six kinds of suggestions were offered and elicited during storyboard presentations. These six suggestions revolved around the following six general questions:

1. What aspects of an algorithm should we illustrate and elide?
2. How should those aspects of the algorithm be represented?
3. What are appropriate sample input data?
4. How should multiple animation views be coordinated?
5. How can a given storyboard feature be implemented in Samba?
6. What is the appropriate scope of our project?

Finally, as discussed above, a minority of students' animations depicted algorithms against the backdrop of a story. As it turned out, storyboard presentation participants took great pleasure in refining and further developing the internal logic and structure of these stories. This was especially true if the stories were creative or innovative, as were the four summarized in Table 1.

During the course of the story-based storyboard presentations, participants sometimes recognized opportunities to revise a scenario so that it better accounted for particular algorithm features, logic, or events of interest. Conversely, and less frequently, participants considered algorithms that might be truer to a given scenario. These discussions accomplished what I have labeled "story tailoring"; see (Hundhausen, 1999, app. B) for a vivid example.

#### 4.2.2 Samba Animation Presentations

As we saw above, students often went to great pains to implement their Samba animations in full detail. However, when it came down to presenting their Samba animations, they tended to fast-forward through much of that detail. Instead, they tended to focus on the same portions of their animations that the storyboards in Study II illustrated exclusively—namely, the interesting events, where something noteworthy or unusual occurred.

Aside from being generally shorter in duration than the storyboard presentations, the Samba presentations differed from the storyboard presentations in three key respects. First, there was noticeably less overall discussion about conceptual issues surrounding algorithms. Instead, Samba presentation sessions tended to be more show-and-tell. Student presenters walked through their animations with minimal interaction with the audience. In fact, prolonged periods of silence, during which the audience and student presenters merely watched the animation, were not uncommon.

Second, when discussions did take place, those discussions focused different topics from those of the storyboard sessions. As discussed above, storyboard presentations were replete with clarifying questions, design discussions, and discussions regarding stories. While Samba presentation sessions often generated clarifying questions, they were generally devoid of discussions regarding design considerations and stories. The exceptions to this were those Samba animation presentations—all of the presentations in Study I, and just a couple in Study II—that did not benefit from a prior storyboard presentation. In such cases, the final presentation sessions served as *de facto* storyboard presentations; the same kinds of design discussions took place. However, since the Samba software was unable to support the kinds of dynamic markup and modification that were commonly used in the storyboard presentations, students and the instructor had to rely more extensively on pointing and gesturing to the screen. In addition, since the Samba software did not support rewinding or backwards execution, animations frequently had to be halted and then restarted in cases in which the audience had questions about an animation event that had already passed by.[9]

Third, whereas storyboard presentation discussions only occasionally addressed implementation *considerations* (one of the suggestion types mentioned above), Samba animation discussions were frequently focused on implementation *details*—how something was implemented, difficulties encountered during implementation, and the like. Occasionally, students vented frustration over the difficulties they encountered in pulling off a given design feature. At other times, such discussions contributed to the "show-and-tell" flavor of the Samba presentation sessions; indeed, in addition to

[9]Interestingly, discussions of stories did not take place during the Samba presentations.

sharing their animations, some students simply felt inspired to share how they had implemented them.

# 5. Discussion

As these observations suggest, the advantage of focusing narrowly on one particular algorithms course is that such a focus enabled me to gain a rich appreciation, informed by multiple actors and perspectives, of AV construction and presentation exercises. The disadvantage is that the observations do not necessarily apply to other algorithms courses taught by other instructors, taken by other students, and offered at other universities. It is important to recognize this limitation in the following discussion, in which I consider, in light of the observations just reported, the three original research questions posed at the beginning of this article. Plainly, great care must be taken in any attempt to generalize the findings I discuss beyond the particular algorithms course I studied.

## 5.1 Do AV construction and presentation assignments promote *relevant* activities?

The observations presented above motivate two insights into the relevance of AV construction and presentation exercises with respect to an undergraduate algorithms course. The first of these is that the relevance of AV construction activities depends intimately both on the method used to perform the construction, and on the requirements to be met by the constructed AV. To see this, one need only consider the stark contrast between students' storyboard construction activities, and their Samba animation implementation activities. By having students construct input-general animations in Samba, Professor Lane and I inadvertently shifted students' focus away from *learning the algorithm*, and towards *learning how to program graphics*. Indeed, rather than concentrating on the high level conceptual issues related to the algorithms they were animating, students became quickly mired in low-level graphics programming issues—for example, how to lay an animation out on the screen, and how to make an animation look clean and polished. Moreover, the difficulties of programming general-purpose graphics to work in the general case—a requirement for the Study I animations—steered students further off course. While perhaps interesting in their own right, these

exercises were clearly peripheral to the concerns of the course. As Professor Lane himself confided, "These are useful exercises, but not in this class."

By contrast, in constructing input-specific, unpolished storyboards, students not only spent substantially less time overall, but the time that they did spend was dedicated to activities that were more relevant to the the course—most notably, research on algorithms, and group discussions about the target algorithms and how best to illustrate them. While storyboard construction necessarily involved implementing an animation, the method of implementation—constructing a presentation with art materials—did not distract students from their focus on algorithms. Further, since the storyboard presentations were understood to be works in progress, and not finished products, students did not while their time away "tweaking" their final storyboards so that they looked polished and presentable. Indeed, the informality of the storyboards appears to have led students to engage in more relevant activities.

The second insight, closely related to the first, concerns the community relevance of the discussions generated by student-constructed AVs: storyboards tended to stimulate more relevant discussions than did Samba animations. My observations suggest two reasons for this. First, because implementing them proved difficult and time-consuming, and because computer science students tend to enjoy sharing implementation stories, Samba-built animations often stimulated more discussion about the effort that went into programming them than about the algorithms that they depicted. By contrast, storyboards required minimal effort to construct, and often appeared rough, scruffy, and unfinished. As a consequence, storyboards invited the criticism and commentary of the audience, which seemed to regard the storyboards as works-in-progress in need of their collaboration.

The second reason for the success of storyboards in stimulating relevant conversations was that they tended to present algorithms at a level of abstraction that was "just right" for the audience. Because they were driven by the algorithms they depicted, Samba animations tended to illustrate an algorithm's procedural behavior in minute detail; no operations could be skipped or elided (although fast-forwarding was possible and used frequently). In contrast, student presenters controlled the

execution of their storyboards. Thus, they could be much more responsive to the requests and questions of the audience. Details that were unimportant to the audience could be skipped or quickly glossed over. Events that were of particular interest and concern could be covered over and over, forwards and backwards, at a slow speed, and even at a different level of detail. In short, because they were student-controlled, storyboards could adapt to the presentation situation, thus serving as a valuable resource for discussions.

## 5.2 Do AV construction and presentation assignments promote *learning*?

According to Social Constructivism, gaining fuller membership within a community fundamentally entails participating within the community in increasingly expert ways. It should be clear that the AV construction and presentation exercises, as they were implemented during my ethnographic fieldwork, gave students an opportunity to engage in activities that are typically performed only by course instructors. As the term progressed, Professor Lane himself came to realize the importance of having students participate as instructors:

> One of the things that I have really come to realize in talking to some of them, giving them an idea of what they should be thinking about animating, is that they become the teacher. So, it's their job to explain *why* an algorithm works, or to show *how* it works. . .The point is, they've got to explain it, and they've got to do it not by standing over somebody and taking questions and answers, but by coming up with this nice video. . .Anytime you're in a situation where you're teaching a subject, you really learn it. And so, this is one of the most [compelling] reasons for [having students construct and present animations].

From John's standpoint as the course instructor, the AV construction exercises had the added benefit of helping him to evaluate students' progress, since they had both to "demonstrat[e] that they know something about an algorithm, and to "prepare[e] a tool where they are teaching what they know." From the students' standpoint, having to construct AVs meant having to take seriously what was important about the algorithms they were learning. Consider, for example, the following interview sequence, in which Mary (M) tells me (E) about the advantage of animation construction, as compared to simply implementing an algorithm:

> M: Well, one specific thing is when we were trying to implement Dijkstra's algorithm, we had to be *aware* of whether or not an edge that had been chosen once was ever going to be chosen again. And, you have to be sure about these sort of things, because if, at some point, it gets unchosen, you have to be able to keep track of that, hold on to the edge, and change the color back, or whatever.

E: Which is not something you'd normally have to do in the course of implementing the algorithm.

M: Right. You wouldn't necessarily see that unless you had actual physical, concrete representation of that edge on the screen, and had to have an actual hold on it. And, alse you had to know is that if it was ever going to be turned on again—you know, be chosen.

Another of my informants, Tom, cited two additional benefits of animation construction as a means of coming to grips with what one has to teach: (a) self-constructed AVs are "objects of [one's] own design," so they make sense to the person doing the construction; and (b) self-constructed AVs require one to engage in a creative process, so that one does not get bored.

The animation presentation sessions also provided students with crucial opportunities to participate as teachers. Indeed, as the description of the animation presentation sessions illustrated, during their presentations sessions, students did many of the things an algorithm teacher would typically do. For example, they provided background on an algorithm; they asked and fielded questions; and they made decisions regarding when it was important to walk slowly through an explanation, and when it was better just to fast-forward. Moreover, as they assumed roles as teachers, students typically received suggestions from Professor Lane and me. Such suggestions gave them essential feedback on how well they were performing as teachers, and on how they might improve their performance.

With respect to the latter, the fact that students' animations were "objects of their own design" seemed crucial to fostering meaningful student-professor interaction in two distinct senses. In one sense, the AV presentation process enabled students to articulate their emerging perspective on an algorithm in the publicly available form of a visual presentation. Through such a presentation, the course instructor could not only identify just how closely that perspective accorded with his own, but also remedy any perceived misconceptions with reference to students' self-constructed AVs. In this sense, students' self-constructed AVs served as *mediational resources* (Roschelle, 1990) that bridged the gap between expert and learner perspectives, enabling mutually meaningful conversations about algorithms to take place.

In another sense, students' self-constructed AVs motivated them to participate more fully in the course. The creative thought that they put into their animation designs, as well as the hard work

that they put into their animation implementations, seemed to vest students in the teaching activities that they were undertaking. Through the process, they began to assume the roles of more central members of the course. In turn, they seemed not only particularly willing to contribute to discussions, but also particularly open and responsive to the feedback they received during their presentation sessions—feedback on how they might "do better" as teachers.

In sum, the AV construction and presentation exercises explored here challenged students to participate, in more expert ways than usual, in the practices of the course in which they were enrolled. My observations suggest that, rather than being put off by this challenge, students were motivated by it. Through constructing and presenting their own AVs, both students' level of competence, and their identity, appeared to be transformed. They began to act, and to see themselves, more as teachers.

## 5.3 How should AV technology be designed so as to support AV construction and presentation?

Ever since Brown's (1988) pioneering work on the BALSA algorithm animation system, interactive AV technology has regarded AV creation and AV viewing as disjoint activities. While AV technology researchers have put considerable effort into making AV creation easier and less time consuming, they have left virtually unexamined the problem of supporting AV presentation. Indeed, Brown's (1988) tape recorder-style interface for interacting with AVs (see Figure 3 for an example) has become the taken-for-granted standard in nearly all subsequent AV technology.

In light of the observations of the student presentation sessions described above, however, AV technologists would do well to reexamine their assumption that AV creation and AV viewing are disjoint activities, as well as their assumption that the AV interaction problem has been solved by a tape recorder-style interface. Those observations suggest that, if AV technology is to truly support student-instructor conversations about algorithms, it must be far more flexible and adaptive than a tape recorder—that, in fact, it must look a lot more like an AV creation environment. Specifically, observations presented above suggest that conversation-supporting AV technology requires three user interface features not supported by extant AV technology.

First, as my above observations indicate, presenters frequently use AVs as a resource for answering questions and requests that emerge out of discussions. To do so, they need to (a) locate quickly a particular point in an AV's execution (in response to a question like "Could you show me the point where the partition element is selected?"); (b) execute the AV in reverse (in response to a question like "That went by too quickly; could we see that section again?"); and (c) vary the speed of the AV's execution (in response to a question like "Could we view that section really slowly?" or "Could you just speed through this section?").

To enable presenters to respond to their audience in these ways, AV technology must support much finer-grained control of AV execution than is supported by current AV technology. In particular, in addition to supporting flexible execution speed (which most extant AV technology already supports), conversation-supporting AV technology must also enable one to execute an AV both forwards and in reverse, and to dynamically set and jump to "visualization points"—points in the AV where events that are interesting to the audience occur.

Second, my observations suggest that conversational participants frequently pointed to and marked up AVs as they were executing. Often they used their fingers or some physical object for pointing. While it is difficult to imagine that a virtual (i.e., computer-based) pointing object would do a better job, a conversation-supporting AV system might consider providing a large, conspicuous mouse pointer for presentations. Moreover, a conversation-supporting AV system should enable one to mark-up, with a virtual pen, an AV as it is executing.

Third, my observations suggest that conversations about AVs frequently give rise to changes in the AV. When students presented storyboards, they could experiment with such changes on the spot by sketching out new visualization objects, or by re-simulating their AV according to the proposed changes. Present AV technology requires considerable reprogramming in order to change an AV. Conversation-supporting AV technology must support an easy means of modifying AVs on the spot, and of trying out the modifications. This suggests that, in the case of conversation-supporting AV technology, the traditional line between AV creation and AV viewing is blurred.

# 6. Related Work

Ethnographic field techniques—including *participant observation*, *interviewing, artifact collection*, *videotaping*, and *keeping fieldnotes*—have a rich tradition of use in a branch of anthropology called *ethnography*, where they have been used to explore and document cultures through in situ field studies (see, e.g., Estroff, 1981; Malinowski, 1922; Mead, 1928; Spradley, 1970).

Far from being the exclusive tools of ethnography, these field techniques have become popular in other disciplines as well, as researchers have become increasingly interested in conducting focused field studies of communities. Designers of computer technology, for example, have recently turned to ethnographic field techniques for help in answering questions about technology design (see, e.g., Anderson, 1994; Blomberg, Giacomi, Mosher, & Swenton-Wall, 1993). In contrast to the ethnographer's goal of attributing culture, the computer technologist's aim, in employing ethnographic field techniques, is to understand how technology fits, or might fit, into the day-to-day practices of a given community of practice, with the ultimate goal of designing usable and useful technology for that community.

Two studies are related to the studies presented here in that they used ethnographic field techniques to study the day-to-day use of visual representations. Bellamy (1994) used both artifact collection and interviews to explore the ways in which experienced programmers use informal notations in their day-to-day programming activities. Likewise, Springmeyer (1992) used participant observation and interviews to study how scientists at a national laboratory use visualization to perform data analysis. She used her observations to develop a taxonomic characterization of the process, as well as to develop a prototype visualization system to better assist scientists in the data analysis process.

Even more closely related to the studies presented here are two studies that used ethnographic field techniques to elicit how computer science students and instructors visualize algorithms. Ford (1993) videotaped students enrolled in his first semester programming class as they sketched visualizations for C++ code fragments. He used the wide range of visualizations generated by his

students as a basis for devising a set of visual program abstractions for aiding students' comprehension of programs (Ford & Tallis., 1993) . In a similar vein, Badre, Baranek, and Morris (1992) used surveys to explore how computer science instructors graphically represent algorithms.

Finally, two field studies of actual computer science courses in which AV technology was used are closely related to the studies presented here. In work that greatly inspired my studies, Stasko (1997) gave similar visualization construction (but not presentation) assignments to students in his junior-level algorithms course. Through questionnaires administered at the end of the course, he learned that students generally enjoyed the assignments and felt that the assignments helped them to better understand the algorithms they visualized. Likewise, in a second-semester programming course, Gurka (1996) relied on student diary keeping to explore students' use of an AV package, as compared to their use of other course resources and activities. She found that student use of AV constituted 35% of their course activity, and that they most often used algorithm visualization to help them complete course programming assignments. The work presented here differs from these studies both in its focus on AV construction and presentation, and in its commitment to using a broad range of ethnographic field techniques to obtain a rich, inside perspective on these activities.

# 7. Conclusions and future work

Inspired by Social Constructivist learning theory, this article has presented a series of ethnographic studies that explored the educational value of AV construction and presentation exercises in an actual undergraduate algorithms course. Two key conclusions can be drawn from the observations made in these studies. First, with proper design and supporting technology, AV construction and presentation exercises can promote course-relevant activities that ultimately contribute to fuller course participation—a Social Constructivist hallmark of learning. As we have seen, when students constructed "low tech," input-specific visualization storyboards instead of "high tech," input-general Samba visualization, not only did they engage in more relevant construction activities, but their ensuing presentations stimulated more relevant discussions about algorithm concepts. Second, there is definite educational value in Social Constructivism's recommended shift in

the ontological status of AV technology—from *knowledge conveyor* to *conversation mediator*. To accommodate such a shift, AV technology ought to enable one to construct and present a visualization just as easily and flexibly as one can do so with conventional art supplies.

The studies presented here set the stage for two main avenues of future research. First, the studies offer preliminary findings that might be validated more rigorously by future research. Take, for example, the finding that storyboard construction exercises enabled students to avoid almost completely the irrelevant low-level implementation details in which students became mired during the Samba AV construction exercises. Plainly, this finding could be verified more rigorously through a more controlled empirical study in which a sample of algorithms students is divided into two groups—Samba and Storyboard— and videotaped as they construct an input general AV (Samba group) or an art supply storyboard (Storyboard group) under closed conditions. Post-hoc Interaction Analysis (Jordan & Henderson, 1995) of the videotapes could be used to classify students' activities; the relevance of each activity could be determined in consultation with one or more algorithms instructors. This would allow one to conduct statistical comparisons of the two groups with respect to the relevance of the activities in which they engaged. In a similar vein, the finding that visualization storyboards stimulate more relevant conversations could be verified simply by adding a presentation activity to the study outlined above.

Systematically verifying the finding that that students become fuller members of a community through their participation in AV construction and presentation exercises poses a methodological challenge. Indeed, how does one measure such slippery notions as a student's level of membership and identify? While I do not claim to have a definitive answer to that question, I can suggest two promising avenues that future research might consider. First, future research could use a carefully-designed *attitude questionnaire* (see, e.g., Shaw & Wright, 1967) to compare the extent to which atudent's' identity changes from the beginning to the end of algorithms courses that include, and do not include, AV construction and presentation exercises. Second, and more ambitious, future research might explore the possibility of using Cultural Consensus Theory (Romney, Weller, &

Batchelder, 1986) as a means of calculating student's level of community membership based on one's ability to construct, and to interpret, AVs in a way that rivals that of a full-fledged course instructor.

The second main avenue of future research motivated by these studies has to do with technology design. The specific research question is, "Can we design computer-based AV technology that supports the creation and presentation of AVs as well as, and preferably better than, simple art supplies?" Based on my ethnographic field studies and prior detailed studies of how students construct and execute visualizations made from simple art supplies (Douglas et al., 1995, 1996), I have explored this question through the development of a prototype sketch-based AV system called ALVIS (Hundhausen & Douglas, 2000a). In ongoing work, I am porting the prototype to a SmartBoard® electronic whiteboard (see http://www.smarttech.com), so that it can better mediate collaborative student-instructor design sessions. My ultimate vision is to use the software as the foundation of an alternative, studio-based approach to teaching algorithms.

## Acknowledgments

This research was conducted as part of my doctoral dissertation (Hundhausen, 1999) in the computer and information science department at the University of Oregon. I gratefully acknowledge the inspiration and astute guidance of my advisor, Sarah Douglas. The instructor and students involved in CIS 315 during my two terms of fieldwork merit many thanks for their willingness to let me observe, videotape, interview, and participate with them.

## References

Anderson, R. J. (1994). Representations and requirements: The value of ethnography in system design. *Human-Computer Interaction, 9*, 151-182.

Badre, A., Baranek, M., Morris, J. M., & Stasko, J. T. (1992). Assessing program visualization systems as instructional aids. In I. Tomek (Ed.), *Computer Assisted Learning, ICCAL '92* (pp. 87-99). New York: Springer-Verlag.

Bellamy, R. K. E. (1994). What does pseudo-code do? A psychological analysis of the use of pseudo-code by experienced programmers. *Human-Computer Interaction, 9*, 225-246.

Blomberg, J., Giacomi, J., Mosher, A., & Swenton-Wall, P. (1993). Ethnographic field methods and their relation to design. In D. Schuler & A. Namioka (Eds.), *Participatory Design: Principles and Practices*. Hillsdale, NJ: Lawrence Erlbaum.

Brown, M. H. (1988). *Algorithm animation.* Cambridge, MA: The MIT Press.

Brown, M. H., & Sedgewick, R. (1984). Progress report: Brown University Instructional Computing Laboratory. *ACM SIGCSE Bulletin, 16*(1), 91-101.

Byrne, M. D., Catrambone, R., & Stasko, J. T. (1999). Evaluating animations as student aids in learning computer algorithms. *Computers & Education, 33*(4), 253-278.

Chaabouni, Z. D. (1996). *A user-centered design of a visualization language for sorting algorithms.* Unpublished Master's Thesis, University of Oregon, Eugene, OR.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms*. Cambridge, MA: The MIT Press.

Crosby, M. E., & Stelovsky, J. (1995). From multimedia instruction to multimedia evaluation. *Journal of Educational Multimedia and Hypermedia, 4*(2/3), 147-162.

Douglas, S. A., Hundhausen, C. D., & McKeown, D. (1995). Toward empirically-based software visualization languages. In *Proceedings of the 11th IEEE Symposium on Visual Languages* (pp. 342-349). Los Alamitos, CA: IEEE Computer Society Press.

Douglas, S. A., Hundhausen, C. D., & McKeown, D. (1996). Exploring human visualization of computer algorithms. In *Proceedings 1996 Graphics Interface Conference* (pp. 9-16). Toronto, CA: Canadian Graphics Society.

Estroff, S. (1981). *Making It Crazy: An Ethnography of Psychiatric Clients in an American Community*. Berkeley: University of California Press.

Ford, L. (1993). *How programmers visualize programs* (Technical report R 271). Exeter, U.K.: Department of Computer Science, University of Exeter.

Ford, L., & Tallis., D. (1993). Interacting visual abstractions of programs. In *Proc. 1993 IEEE Workshop on Visual Languages* (pp. 93-97). Los Alamitos, CA: IEEE Comptuer Society Press.

Gurka, J. S. (1996). *Pedagogic Aspects of Algorithm Animation.* Unpublished Ph.D. Dissertation, University of Colorado, Boulder, CO.

Hansen, S. R., Narayanan, N. H., & Schrimpsher, D. (2000). Helping learners visualize and comprehend algorithms. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning, 1*(1).

Hundhausen, C. D. (1999). *Toward effective algorithm visualization artifacts: Designing for participation and communication in an undergraduate algorithms course.* Unpublished Ph.D. Dissertation, University of Oregon, Eugene, OR.

Hundhausen, C. D., & Douglas, S. A. (2000a). Shifting from "high fidelity" to "low fidelity" algorithm visualization technology. In *SIGCHI 2000 Extended Abstracts: Conference on Human Factors in Computing Systems* (pp. 179-180). New York: ACM Press.

Hundhausen, C. D., & Douglas, S. A. (2000b). Using visualizations to learn algorithms: Should students construct their own, or view an expert's? In *Proceedings 2000 IEEE International Symposium on Visual Languages* (pp. 21-28). Los Alamitos: IEEE Computer Society Press.

Jarc, D. J., Feldman, M. B., & Heller, R. S. (2000). Assessing the benefits of interactive prediction using web-based algorithm animation courseware. In *Proceedings SIGCSE 2000* (pp. 377-381). New York: ACM Press.

Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *Journal of the Learning Sciences, 4*(1), 39-103.

Kann, C., Lindeman, R. W., & Heller, R. (1997). Integrating algorithm animation into a learning environment. *Computers & Education, 28*(4), 223-228.

Knuth, D. E. (1973). *The Art of Computer Programming*. Menlo Park, CA: Addison-Wesley.

Lave, J., & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation.* New York: Cambridge University Press.

Lawrence, A. W. (1993). *Empirical studies of the value of algorithm animation in algorithm understanding.* Unpublished Ph.D. dissertation, Georgia Institute of Technology, Atlanta.

Malinowski, B. (1922). *Argonauts of the Western Pacific: An Account of Native Enterprise and Adventure in the Archipelagoes of Melanesian New Guinea*. New York: Dutton.

Mead, M. (1928). *Coming of Age in Somoa.* New York: Blue Ribbon Books.

Naps, T. (1990). Algorithm visualization in computer science laboratories. In *Proceedings of the 21st SIGCSE Technical Symposium on Computer Science Education* (pp. 105-110). New York: ACM Press.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

Roman, G. C., Cox, K. C., Wilcox, C. D., & Plun, J. Y. (1992). Pavane: A system for declarative visualization of concurrent computations. *Journal of visual languages and computing, 3*(2), 161–193.

Roschelle, J. (1990). *Designing for conversations.* Paper presented at the AAAI Symposium on Knowledge-Based Environments for Learning and Teaching, Stanford, CA.

Shaw, M. E., & Wright, J. M. (1967). *Sacles for the Measurement of Attitudes.* San Francisco: McGraw-Hill.

Spradley, J. P. (1970). *You Owe Yourself a Drunk: An Ethnography of Urban Nomads*. Boston: Little, Brown.

Springmeyer, R. R. (1992). *Designing for scientific data analysis: From practice to prototype.* Unpublished Ph.D. dissertation, University of California-Davis, Davis, CA.

Stasko, J., Badre, A., & Lewis, C. (1993). Do Algorithm Animations Assist Learning? An Empirical Study and Analysis. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems* (pp. 61-66). New York: ACM Press.

Stasko, J., & Kraemer, E. (1993). A methodology for building application-specific visualizations of parallel programs. *Journal of parallel and distributed computing, 18*(2), 258–264.

Stasko, J. T. (1990). TANGO: A framework and system for algorithm animation. *IEEE Computer, 23*(9), 27-39.

Stasko, J. T. (1997). Using student-built animations as learning aids. In *Proceedings of the ACM Technical Symposium on Computer Science Education* (pp. 25-29). New York: ACM Press.

# Figures

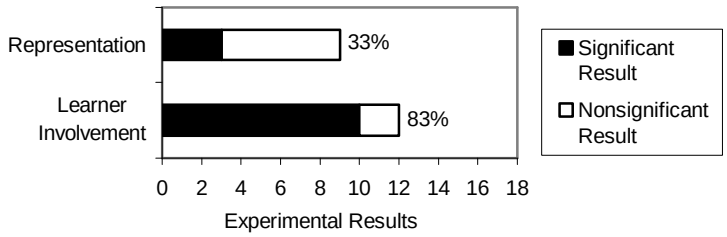


Figure 1. Results of AV effectiveness experiments broadly classified by their independent variables

(a) Samba Script Fragment

```
view Freqs
% start building frequency graph
{
flextext jab1 .5 0.7605 0 white screen-bold-14 j
flextext jab2 .515 0.7605 0 white screen-bold-14 a
flextext jab3 .53 0.7605 0 white screen-bold-14 b
flextext jab5 .56 0.7605 0 white screen-bold-14 e
flextext jab6 .575 0.7605 0 white screen-bold-14 r
flextext jab7 .590 0.7605 0 white screen-bold-14 w
flextext jab8 .605 0.7605 0 white screen-bold-14 o
flextext jab9 .62 0.7605 0 white screen-bold-14 c
flextext jab10 .635 0.7605 0 white screen-bold-14 k
flextext jab11 .65 0.7605 0 white screen-bold-14 y

flextext twas1 .5 0.7 0 white screen-bold-14 t
flextext twas2 .515 0.7 0 white screen-bold-14 w
flextext twas3 .53 0.7 0 white screen-bold-14 a
flextext twas4 .545 0.7 0 white screen-bold-14 s

flextext bril1 .57 0.7 0 white screen-bold-14 b
flextext bril2 .585 0.7 0 white screen-bold-14 r
flextext bril3 .595 0.7 0 white screen-bold-14 i
flextext bril4 .605 0.7 0 white screen-bold-14 l
flextext bril5 .615 0.7 0 white screen-bold-14 l
flextext bril6 .625 0.7 0 white screen-bold-14 i
flextext bril7 .64 0.7 0 white screen-bold-14 g
. . .
```

(b) Samba Animation Snapshot

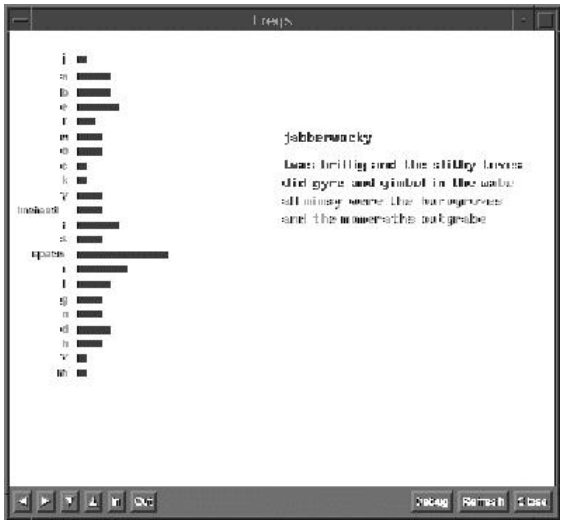


Figure 2. Sample Samba Script Fragment and Animation Snapshot


Figure 3. The Polka Control Panel

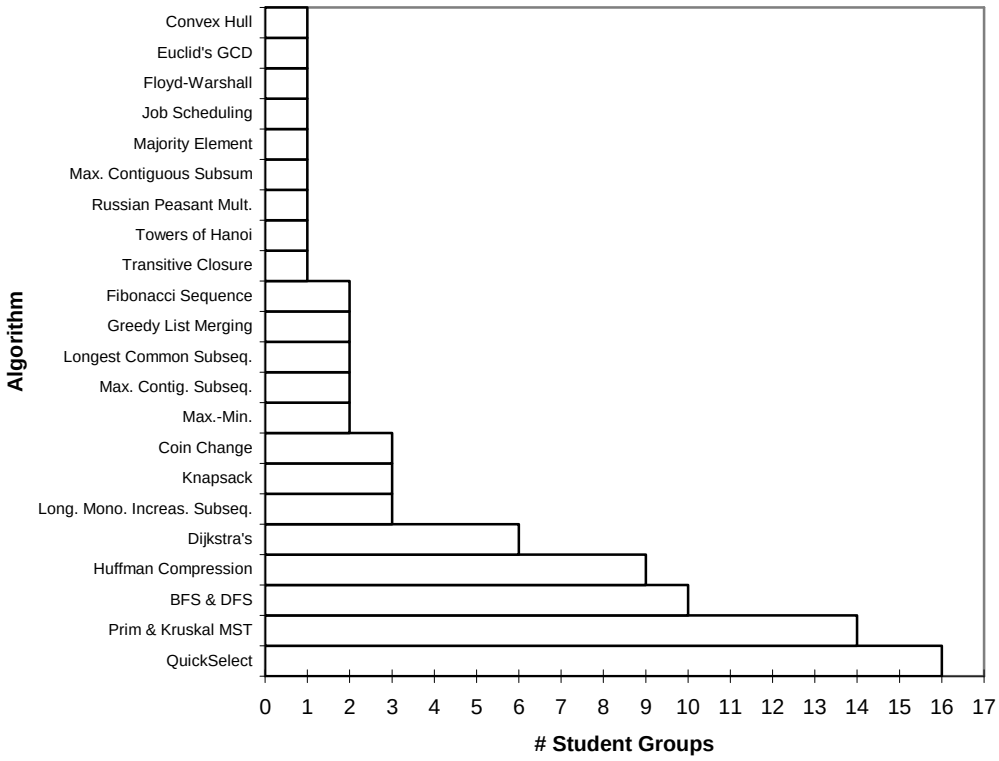


Figure 4. Algorithm Themes Animated by Student Groups

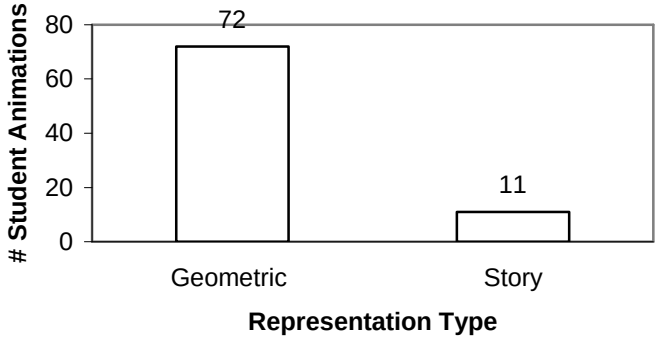


Figure 5. Number of Geometric and Story-based Animations

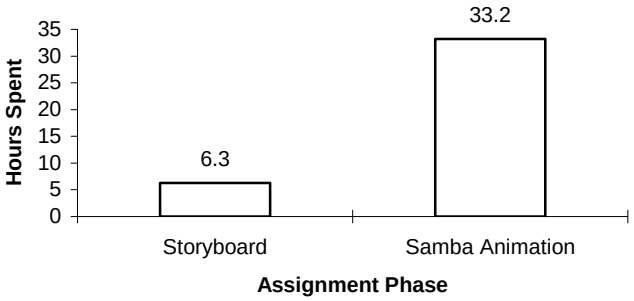


Figure 6. Time Spent on Storyboards and Samba Animations

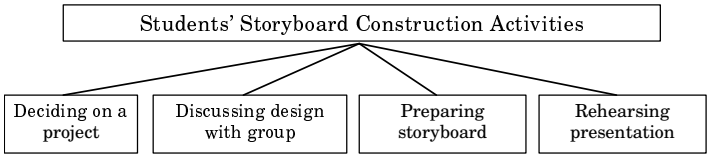


Figure 7. Students' Storyboard Construction Activities

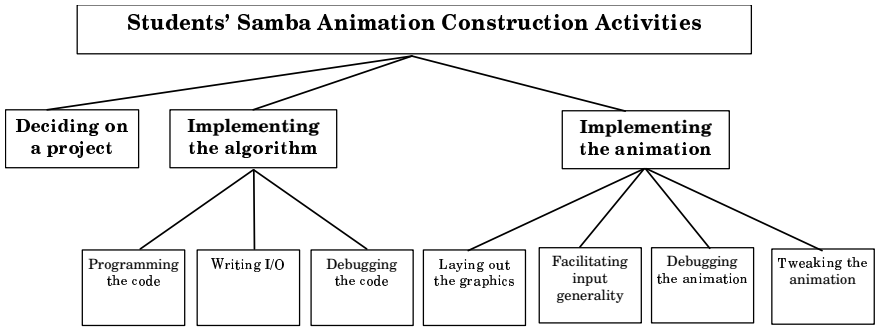


Figure 8. Students' Samba Animation Construction Activities

# Tables

Table 1. Four Storyboard Stories That Stood Out For Their Creativity and Innovation

| Algorithm(s) | Story Title | Synopsis |
|---|---|---|
| Breadth First Search vs. Depth-First Search | Wizard vs. Scientist | A wizard with the ability to teleport is pitted against a scientist with the ability to clone himself. The two are challenged to find their way out of a maze using their supernatural abilities. Who will win? |
| QuickSelect Algorithm | Select Mining Corporation | You are charged with the task of improving the efficiency of mining operations at the Select Mining Corporation. Mined nuggets move along on a conveyor belt. Only the nth heaviest nugget in a given batch is to be selected. How can the Select Mining Corporation select it most efficiently? |
| Floyd-Warshall Algorithm | Matt the Pilot | Your brother, Matt, is a retired military pilot who wants to make some extra money while he travels the world. Between some cities, he can fly a military plane and make money. Between other cities, he must fly with a commercial carrier and lose money. Given a beginning city and a destination, what route should he take to maximize his profit? |
| Dijkstra's algorithm + a greedy selection algorithm | Field Trip to Baker City | Professor Midas decides to load his algorithms class onto his psychedelic bus for a field trip to the Computer Science Museum in Baker City, Oregon. What is the shortest route from the University of Oregon (Eugene, Oregon) to Baker City, and how can the trip be made with the fewest gas stops? |