# TOWARD EFFECTIVE ALGORITHM VISUALIZATION ARTIFACTS: DESIGNING FOR PARTICIPATION AND COMMUNICATION IN AN UNDERGRADUATE ALGORITHMS COURSE

by

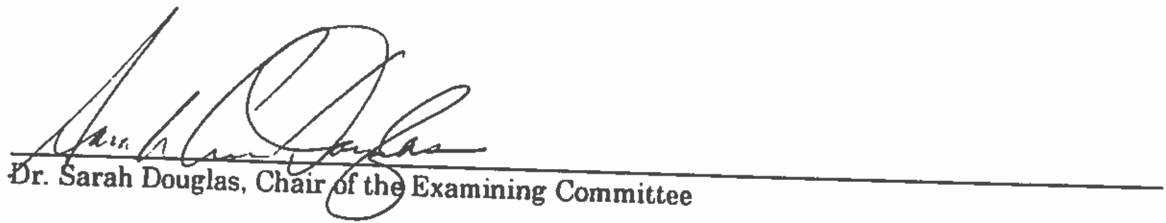
CHRISTOPHER DAVID HUNDHAUSEN

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

June 1999

"Toward Effective Algorithm Visualization Artifacts: Designing for Participation and Communication in an Undergraduate Algorithms Course," a dissertation prepared by Christopher D. Hundhausen in partial fulfillment of the requirements for the Doctor of Philosophy Degree in the Department of Computer and Information Science. This dissertation has been approved and accepted by:



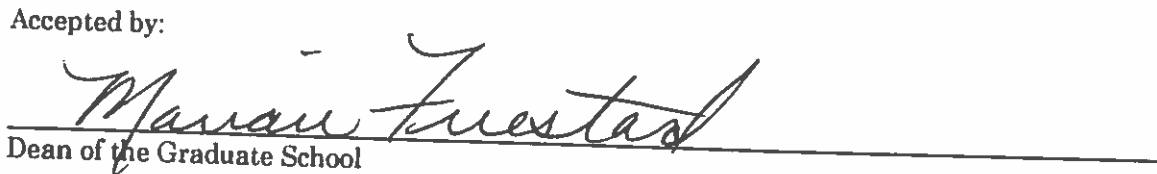
Dr. Sarah Douglas, Chair of the Examining Committee

May 17, 1999
Date

Committee in charge:

Dr. Sarah Douglas, Chair
Dr. Christopher Wilson
Dr. Michal Young
Dr. Harry Wolcott
Dr. John Stasko

Accepted by:



Dean of the Graduate School

An Abstract of the Dissertation of

Christopher David Hundhausen for the degree of Doctor of Philosophy

in the Department of Computer and Information Science to be taken June 1999

Title: TOWARD EFFECTIVE ALGORITHM VISUALIZATION ARTIFACTS: DESIGNING FOR PARTICIPATION AND COMMUNICATION IN AN UNDERGRADUATE ALGORITHMS COURSE

Approved:
Dr. Sarah Douglas

Algorithm visualization (AV) software graphically illustrates how computer algorithms work. While the software initially had much promise as a pedagogical aid, research studies designed to substantiate its pedagogical benefits have yielded markedly mixed results. I argue that to harness the pedagogical promise of AV software, we need to rethink the theory of effectiveness that has guided its design and pedagogical use. My starting point is an alternative theoretical foundation that views learning not at the level of the individual, but rather at the level of the community of practice. On this alternative view, learning is seen in terms of participating more centrally in the practices of the community. To tailor this theoretical perspective to the particulars of the community of practice in which algorithms learning takes place, I conducted an ethnographic study of an undergraduate algorithms course in which AV software was used to facilitate students' more central participation in the community. Specifically, students were asked to use AV software to construct and present their own visualizations—two activities commonly performed only by

community experts (algorithms instructors). The key finding of the study is that requiring students to use conventional AV software in this way actually impedes learning within the community, because it requires students to put inordinate amounts of time into community-irrelevant activities, and because it discourages students and instructors from engaging in meaningful conversations about algorithms. On the other hand, asking students to construct and present homemade visualizations made out of simple art supplies appears to avoid these problems. To explore this finding further, this dissertation pursues two parallel research directions: (1) a controlled experiment that tests the hypothesis that, on a test of procedural understanding and recall, students who construct their own, homemade visualizations will outperform students who interact with a visualization constructed by an expert; and (2) a prototype AV system that supports the construction and presentation of unpolished, pen-and-paper visualizations. This research provides the beginnings of an alternative theory of effectiveness, which emphasizes the importance of students' constructing and discussing unpolished, pen-and-paper visualizations as a means of participating in a community of practice.

# CURRICULUM VITA

NAME OF AUTHOR: Christopher David Hundhausen

PLACE OF BIRTH: Madison, Wisconsin

DATE OF BIRTH: September 24, 1969

## GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon
Lawrence University

## DEGREES AWARDED:

Doctor of Philosophy in Computer and Information Science, 1999, University of Oregon
Master of Science in Computer and Information Science, 1993, University of Oregon
Bachelor of Arts in Math/Computer Science, 1991, Lawrence University

## AREAS OF SPECIAL INTEREST:

Human-Computer Interaction
Computer-Supported Collaborative Learning

## PROFESSIONAL EXPERIENCE:

Teaching Assistant and Instructor, Department of Computer and Information Science, University of Oregon, Eugene, 1995-99

Usability Engineer, Development Tools Group, Microsoft Corporation, Redmond, Washington, 1996-97

Research Assistant, Department of Computer and Information Science, University of Oregon, Eugene, 1994-95

Administrative Programmer, Graduate School, University of Oregon, Eugene, 1991-93

## AWARDS AND HONORS:

Participant, Doctoral Consortium of the Association for Computing Machinery's Special Interest Group on Computer-Human Interaction, 1998

Andrew C. Berry-James Stewart Prize in Mathematics (awarded to outstanding graduating senior in mathematics department), Lawrence University, 1991

Phi Beta Kappa, Lawrence University, 1990

Mortar Board, Lawrence University, 1990

## GRANTS AND SCHOLARSHIPS:

Mary Chambers Brockelbank Scholarship, University of Oregon, 1999

Graduate Student Research Award, University of Oregon, 1996

Fulbright Scholarship, University of Karlsruhe, Germany, 1993-94

Cray Research Grant, 1988-91

Valedictorian Scholarship, Lawrence University, 1987-91

## PUBLICATIONS:

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

Douglas, S.A., McKeown, D., & Hundhausen, C.D. (1996). Exploring human visualization of computer algorithms. In *Proceedings Graphics Interface '96.* Toronto, Canada: Canadian Information Processing Society.

Hundhausen, C. D. (1998). Toward Effective Algorithm Visualization Artifacts: Designing for Participation and Negotiation in an Undergraduate Algorithms Course. In *Human Factors in Computing Systems: CHI 98 Summary* (pp. 54-55). New York: ACM Press.

Naps, T.L., & Hundhausen, C.D. (1991). The evolution of an algorithm visualization system. In *Proceedings of the 24th Annual Small College Computing Symposium* (pp. 259–266), Morris, MN.

ACKNOWLEDGEMENTS

This epic journey would not have been possible without the help of several individuals. To begin with, I owe a big hug and kiss to my parents, Patricia and David Hundhausen, who have always been remarkably supportive of my interests, despite the fact that those interests have not always been understandable or interesting to them.

Lee Schmidt, my high school mentor, and Tom Naps, my undergraduate mentor, planted the initial seeds for my venture into computer science and research. Lee sparked my interest in computer programming in Pascal, while Tom sparked my interest in doing long-term research, and then patiently taught me how to do it during the four summers I helped him to build software for Lawrence University's algorithm visualization laboratory (NSF ILI grant #88-51781).

I am grateful to Walter Tichy for inviting me to pursue the preliminary stages of this research as a Fulbright Scholar in his Institute for Programming Structures and Data Organization at the University of Karlsruhe, Germany. During my "Traumjahr" there (1993-94), Arne Frick, Stefan Hänssgen, Rick Chamberlin, Delel Chaabouni, Christoph Jerger, Aida Chouchane, Karim Diab, Frederic Million, Fabien Fersing, and Richard Kupprion sustained me with much needed encouragement and friendship as I learned to live "auf die echte deutsche Weise" (in the true German way). When I arrived back in Eugene, Dr. Allen Malony advised and funded me during my first year in the Ph.D. program, while Gunnar Sacher helped me to maintain balance by beckoning me away from my computer for hiking and camping trips.

The ethnographic fieldwork and experiment presented in this dissertation would not have been possible without the enthusiastic participation of numerous CIS 315 students and

their CIS 315 instructor; unfortunately, I cannot mention them by name. I am grateful to the University of Oregon's College of Arts and Sciences and Graduate School for two grants (the Mary Chambers Brockelbank Scholarship, and a Graduate Student Research Award) that paid the participants in the experiment presented in Chapter VI. The prototype language and system presented in this dissertation would not have been possible without the help of (a) Hank Bennett, who programmed the back-end animation engine and a significant portion of the user interface, and (b) fellow HCI lab member Ted Kirkpatrick, who furnished ingenious advice at times when the Microsoft® Foundation Classes had me really stumped. Ted also kept me honest during the statistical analysis of my experiment's data.

I am grateful to Harry Wolcott for getting me excited about ethnography, and for offering sage advice on the ethnographic research presented in this dissertation. I am grateful to John Stasko, one of the pioneers of algorithm visualization, for being such a wonderful remote collaborator and discussant throughout my graduate studies, and for serving on my committee. Last, but not least, I am deeply indebted to my inspirational advisor, Sarah Douglas, who challenged me while remaining encouraging all the way through, especially in the final months, when I needed her support the most.

My greatest thanks, however, goes lovingly to my best friend, life partner, and fiancée, Laura Girardeau, who has inspired me to grow in ways I could never have imagined. Despite my tendency to act as though algorithm visualization is the most important thing in life, she was able to see through to my true self, remaining passionate, patient, giving, and forgiving. Namaste!

DEDICATION

To my old self, who thought that he had to achieve to be loved

TABLE OF CONTENTS


Chapter Page

I. INTRODUCTION ..... 1

Problems ..... 5
Paths to Solutions ..... 7
The Thesis ..... 9
Brief Summary of the Dissertation ..... 10
EF Theory, Its Stronghold on Past Research, and Its Inadequacy ..... 10
Choosing an Alternative Theory ..... 12
Empirically Exploring the Sociocultural Constructivist Approach ..... 12
A Framework of Cause and Effect ..... 14
Two Research Directions ..... 15
Organization of the Dissertation ..... 22

II. EPISTEMIC FIDELITY THEORY ..... 23

The Allegory of Musica ..... 24
Key Assumptions ..... 26
The Knowledge Representation Assumption ..... 27
The Knowledge Flow Assumption ..... 27
The Graphical Medium Effectiveness Assumption ..... 28
Overriding Assumption: Epistemic Fidelity Is Crucial ..... 28
Practical Implications ..... 29
Variations on EF Theory ..... 30
Weak EF Theory (Learner Involvement) ..... 31
Weak EF Theory (Individual Differences) ..... 32
Weak EF Theory (Dual-Coding) ..... 32
EF Theory's Influence on Past Research ..... 32
Influence on AV Technology Design ..... 33
Support for Knowledge Flow ..... 34
Support for Algorithmic Fidelity ..... 38
Support for Cultural Fidelity ..... 42
Influence on Effectiveness Evaluation ..... 43
Influence on Pedagogy ..... 45
AV in Lectures ..... 46
AV in Labs ..... 47
AV for Individual Study ..... 48
A Critique of EF Theory ..... 49
Summary ..... 55

Page

III. CONSTRUCTIVISM: AN ALTERNATIVE THEORETICAL FOUNDATION....... 57

Cognitive Constructivism .... 59
Key Assumptions.... 60
Learners Actively Construct Their Own Knowledge.... 60
Stages of Intellectual Development: Concrete to Abstract .... 60
Implications for Pedagogy .... 61
Implications for Technology Design.... 63
Implications for Effectiveness Evaluation.... 64
Discussion: Algorithms as a Social Practice .... 66
Sociocultural Constructivism .... 69
The Allegory of Musiphonia .... 70
Key Assumptions.... 72
Knowledge and Learning within a Community.... 72
Learning Is Changing Participation and Identity .... 73
Access Is Crucial .... 75
Implications for Pedagogy .... 76
Implications for Technology Design.... 79
Implications for Effectiveness Evaluation.... 81
Summary and Research Questions .... 82

IV. ETHNOGRAPHIC STUDIES .... 84

Background.... 87
Informants: Students and the Instructor .... 88
Algorithm Visualization Assignments and Supporting Technology.... 88
Study I Visualization Assignments .... 89
Study II Visualization Assignments.... 91
Field Techniques .... 91
Observations.... 93
Animation Construction .... 96
Storyboard Construction Activities .... 97
Samba Construction Activities .... 98
Animation Presentation and Discussion .... 102
Storyboard Presentations.... 102
Samba Animation Presentations .... 105
Discussion.... 107
Activity Relevance.... 107
Designing for Conversations about Algorithms .... 110
AV Construction and Presentation as Expert Forms of COSA
Participation.... 112
Summary.... 114

Page

V. A FRAMEWORK OF CAUSE AND EFFECT ........ 116

AV Effectiveness Factors ........ 118
Representation Factors ........ 118
Input Generality and Typset Fidelity ........ 119
Story Content ........ 120
Graphics Creation Technique ........ 121
Self-Construction ........ 122
Instructor Communication ........ 123
AV Effectiveness Measures ........ 124
Activity Relevance ........ 125
Understanding and Recall ........ 125
Effective Communication ........ 128
Community-Building ........ 129
Hypotheses ........ 131
The Activity Relevance Hypothesis ........ 132
The Communication Effectiveness Hypothesis ........ 133
The Understanding and Recall Hypothesis ........ 134
The Community-Building Hypothesis ........ 137
Summary and Research Directions ........ 139

VI. EXPERIMENTAL STUDIES ........ 142

A Series of Experiments ........ 143
Experiment 1: Active Viewing versus Self-Construction ........ 145
Design ........ 146
Equalizing the Treatments ........ 147
Participants ........ 149
Materials ........ 150
Active Viewing Treatment Group ........ 151
Self Construction Treatment Group ........ 154
Tasks ........ 156
Procedure ........ 159
Scoring the Traces and Programs ........ 159
Results ........ 161
Discussion ........ 164
Summary ........ 168

VII. PROTOTYPE LANGUAGE AND SYSTEM ........ 171

Empirical Foundations ........ 174
AV Content ........ 175
Process of AV Construction ........ 176
Process of AV Execution and Presentation ........ 177
Design Requirements ........ 178
SALSA ........ 180
Data Types ........ 180
Spatial Relations ........ 182
Commands ........ 183

Page

ALVIS ........ 184
Overview ........ 185
Storyboard Construction Interface ........ 188
Creating and Placing Storyboard Elements ........ 189
Animating Cutouts ........ 191
Storyboard Presentation Interface ........ 193
Example of the Prototype in Use ........ 194
Creating the Storyboard Elements ........ 196
Placing the Storyboard Elements ........ 198
Programming the Spatial Logic ........ 200
Presenting the Storyboard ........ 204
Related Work ........ 207
Summary ........ 210

VIII. CONCLUSION ........ 212

Research Contributions ........ 216
Directions for Future Research ........ 218
The Understanding and Recall Hypothesis ........ 219
The Activity Relevance Hypothesis ........ 220
The Communication Effectiveness Hypothesis ........ 221
The Community-Building Hypothesis ........ 222
The Prototype Language and System ........ 229
A Vision for the Future: The "Algorithms Studio" ........ 230

APPENDIX

A. ETHNOGRAPHIC STUDY I ........ 233

Background ........ 234
Getting in the door ........ 235
Disclaimers ........ 235
Using Samba for Algorithm Animation ........ 236
Field Techniques ........ 237
"Experiencing" Techniques ........ 238
"Enquiring" Techniques ........ 239
"Examining" Techniques ........ 240
Informants: Students and the Instructor ........ 240
Study I Animation Assignments ........ 241
How Students Went about the Animation Assignments ........ 242
Time Spent ........ 243
Animation-Building Activities ........ 243
Deciding on a Project ........ 244
Implementing the Algorithm ........ 244
Programming the Animation ........ 245
The Animation Presentation Sessions ........ 246
A Typical Presentation Session ........ 246
Assignment Improvement Discussions ........ 249
The Value of Programming the Algorithm ........ 249

Page

The Difficulties of Graphics Layout ..... 250
Input Generality a Formidable Challenge ..... 251
The Instructional Side of the AV Assignments ..... 252
Assessing the Costs and Benefits of AV Assignments ..... 253
Students' Perspective ..... 253
The Professor's Perspective ..... 254
Grading is Difficult and Takes Too Long ..... 255
Students' Time Not Well-Spent on the Animation Assignments ..... 255
Why are AV Assignments Effective? ..... 257
Discussion ..... 259

B. ETHNOGRAPHIC STUDY II ..... 261

Study II Animation Assignments ..... 262
Research questions ..... 263
Field techniques ..... 264
Students' Work on the Revised Animation Assignment ..... 264
Time Spent ..... 265
Animation-Building Activities ..... 266
Approaches to the Assignment ..... 267
The Presentation Sessions ..... 267
Storyboard presentations ..... 268
Clarifying Questions ..... 269
Design Discussions ..... 269
Tailoring Scenarios to Algorithms, and Algorithms to Scenarios ..... 272
Final Animation Presentations ..... 276
Was the Revised Animation Assignment Format an Improvement? ..... 278
Did a Single Animation Assignment Work out Better? ..... 278
Was the Absence of an Input Generality Requirement an Improvement? .. 279
Assessing the Storyboard Phase of the Assignment ..... 280
Comparing the Storyboard Phase to the Final Animation Phase ..... 281
Differences in What Students Learned ..... 282
Relevance of What Students Learned ..... 283
Are Both Phases Necessary? ..... 284
Discussion ..... 285

C. ETHNOGRAPHIC STUDY MATERIALS ..... 286

CIS 315 Syllabus for Study I ..... 287
CIS 315 Syllabus for Study II ..... 290
Study I Animation Assignments ..... 291
Assignment 1 ..... 291
Assignment 2 ..... 291
Assignment 3 ..... 292
Calls for Participation ..... 294
Informed Consent Agreements ..... 295
Study I E-Mail Surveys ..... 300
Study II Animation Assignment ..... 302
Study II: Guidelines for Developing a Storyboard Presentation ..... 304

Page

Study II Diary Form .......... 305

D. EXPERIMENT MATERIALS .......... 306

Preliminary Interest Questionnaire .......... 307
Background Information Questionnaire .......... 308
Informed Consent Agreement .......... 309
Instructions: Active Viewing Group .......... 311
Instructions: Self-Construction Group .......... 317
Tracing Task Instructions .......... 323
Tracing Tasks .......... 325
Programming Task Instructions .......... 329
Java QuickSelect Skeleton .......... 331
C++ QuickSelect Skeleton .......... 334
Instructions for Finishing Up .......... 337
Exit Questionnaire: Active Viewing Group .......... 338
Exit Questionnaire: Self-Construction Group .......... 339

E. EXPERIMENT DATA .......... 340

F. SALSA LANGUAGE SUMMARY .......... 342

Commands .......... 343
Storyboard Element Creation, Deletion, and Attribute Modification .......... 343
Conditionals and Iteration .......... 344
Animation .......... 346
Attributes of Storyboard Elements .......... 348
SALSA Object Attributes .......... 348
Cutout attributes .......... 349
Position Attributes .......... 349
Grid Attributes .......... 349

G. SAMPLE QUESTIONNAIRE FOR ASSESSING STUDENT ATTITUDES TOWARD AN ALGORITHMS COURSE .......... 351

BIBLIOGRAPHY .......... 355

## LIST OF TABLES


Table Page

1. Summary of Ten AV Effectiveness Experiments ........ 50
2. Independent Variables Manipulated in Ten AV Effectiveness Experiments ........ 51
3. Ten AV Effectiveness Experiments vis-à-vis the Version of EF Theory For Which They Were Designed to Provide Evidence ........ 52
4. Four Storyboard Stories That Stood Out For Their Creativity and Innovation ........ 105
5. Independent and Dependent Variables in the Understanding and Recall Hypothesis ........ 144
6. Series of Planned Experiments ........ 144
7. Design of Experiment 1 ........ 146
8. The Table Participants Filled in for Each Trace ........ 158
9. Mean Trace Scores and Times of the Two Treatments ........ 162
10. Mean Program Scores and Times of the Two Treatments ........ 163
11. Implications of Framework Hypotheses for AV Technology Design ........ 172
12. The Ten Spatial Relations Supported By SALSA ........ 183
13. Brief Summary of SALSA Commands ........ 184
14. A 2 × 2 Design that Explores Self Construction and Instructor Communication ........ 220
15. Ethnographic Field Techniques vis-à-vis Wolcott's Taxonomy. ........ 238
16. Excerpts from the Content Log of a Typical Presentation ........ 248
17. Field Techniques Used to Address Study II's Research Questions ........ 264
18. The Six Kinds of Questions That Were Offered and Elicited During Storyboard Presentations ........ 270
19. Five Storyboard Stories that Stood Out for Their Creativity and Innovation. ........ 273
20. Curt and Jeff's Final Animation Scenario ........ 277

# LIST OF FIGURES


Figure Page

1. Knuth's Illustration of the Deque as a Railway Switching Network ........ 2
2. A snapshot of the "Grand Race" in *Sorting Out Sorting* ........ 3
3. A Snapshot from An Interactive Session with BALSA ........ 4
4. Graphical Summary of the Dissertation ........ 11
5. Pseudocode Description of Insertion Sort Algorithm ........ 20
6. An AV of the Insertion Sort Algorithm ........ 20
7. The POLKA User Interface for Controlling Animation Execution ........ 22
8. The Musibeest ........ 25
9. A Schematic Diagram of Knowledge Flow According to EF Theory ........ 28
10. Knowledge Flow in Brown's (1988) User Model for AV Software ........ 36
11. Prototypical Design of an AV Effectiveness Experiment ........ 45
12. Summary of the Experimental Support for each Version of EF Theory ........ 53
13. The Historical Evolution of AV Effectiveness Experiments and Empirical Studies ........ 55
14. Layers of AV Technology-Mediated Participation ........ 78
15. Sample Samba Script Fragment and Animation Snapshot ........ 90
16. The Polka Control Panel ........ 90
17. Algorithm Themes Animated by Student Groups ........ 94
18. Number of Geometric and Story-Based Animations ........ 94
19. Time Spent on Storyboards and Samba Animations ........ 97
20. Students' Storyboard Construction Activities ........ 98
21. Students' Samba Animation Construction Activities ........ 99
22. A Taxonomy of AV Effectiveness Factors ........ 118
23. A Taxonomy of AV Effectiveness Measures ........ 124

Page

24. Graphical Summary of the Framework ........ 132
25. The Activity Relevance Hypothesis ........ 132
26. The Communication Effectiveness Hypothesis ........ 133
27. The Understanding and Recall Hypothesis ........ 135
28. The Community-Building Hypothesis ........ 137
29. The Understanding and Recall Hypothesis ........ 143
30. The “Console” Window ........ 151
31. The Polka Control Panel ........ 152
32. The “Forest Rangers” Window ........ 153
33. The “History” Window ........ 155
34. The “History Legend” Window ........ 155
35. Box Plot of Trace Scores of Each Condition ........ 163
36. Box Plot of Trace Times of Each Condition ........ 163
37. Box Plot of Programming Task Scores ........ 164
38. Box Plot of Programming Task Time ........ 164
39. Predefined Cutout Refpoints ........ 181
40. Cutouts in a Sample SALSA Storyboard ........ 182
41. Snapshot of the ALVIS Environment ........ 186
42. The ALVIS Menus ........ 188
43. The Create Cutout Dialog Box ........ 190
44. The Cutout Graphics Editor ........ 191
45. The Script Move Dialog Box ........ 192
46. The Execution Control Toolbar ........ 194
47. The Present/Edit Toolbar ........ 194
48. Pseudocode Description of the Bubble Sort Algorithm ........ 195
49. The Football Bubble Sort Storyboard ........ 196
50. The ALVIS Environment After Creating First Player ........ 197

Page

51. Creating a Grid with the Create Grid Dialog Box ........ 199

52. The ALVIS Environment After All Cutouts Have Been Placed ........ 200

53. SALSA Script for Football Bubble Sort Storyboard ........ 204

54. The ALVIS Environment Before the Players Are Swapped ........ 206

55. "Attempt to Edit Previously Executed Line" Dialog Box ........ 206

56. A Taxonomy of AV Construction Techniques ........ 208

57. Example of the Semantic-Level Analysis Technique ........ 227

58. A Schematic of the Area in Which the Fieldwork Was Conducted ........ 237

59. The Assignment 2 Animation Projects ........ 242

60. The Assignment 3 Animation Projects ........ 243

61. A Taxonomy of How Students Spent Their Time on Animation Assignments ........ 244

62. The Animation Projects Chosen By the 28 Student Groups ........ 265

63. Comparison of Time Students Spent on the Storyboard and Animation Implementation Components of the Assignment ........ 266

64. The Number of Student Groups Adopting Five Alternative Animation Implementation Strategies. ........ 267

# CHAPTER I

# INTRODUCTION

> One's object is then to have a clear mental picture of the state of the machine at each moment in the computation. This object can only be achieved with a struggle.
> (Turing, 1950, p. 459)

Algorithms, the building blocks of computer software, are inherently abstract entities. Indeed, aside from the electronic pulses that flow through the computer as they execute, they lack a concrete representation in the natural world. Their lack of a tangible real-world representation distinguishes them from other objects of scientific inquiry, such as clouds, wildebeests, and amoebas; it also makes them notoriously difficult to teach and learn.

As anyone who has ever had to explain an algorithm to someone else will attest, the use of graphical notations in the explanation of algorithms seems both extremely natural and particularly appropriate. A survey of the pedagogical literature and algorithms research bears this out; ever since the publication of the first volumes of Knuth's seminal series *The Art of Computer Programming* (Knuth, 1973), graphical illustrations of algorithms have become commonplace as visual aids in computer science textbooks and journal articles.

Consider, for instance, Knuth's illustration of a data structure called the *deque*.[1] A generalization of stacks and queues, the deque is a linear list in which all insertions and deletions are done at the ends of the list. Inspired by E.W. Dijkstra's illustrations of stacks, Knuth (1973) depicts the deque as a railroad switching network (see Figure 1). Railroad cars to be added to the deque must pass through a switch; depending on how the switch is set,

---

[1]A double-ended queue, pronounced "deck."

cars roll either directly to the head of the deque, or around a loop to the tail of the deque. Likewise, depending on how the switches are set, railroad cars may be removed either from the head or tail of the deque.

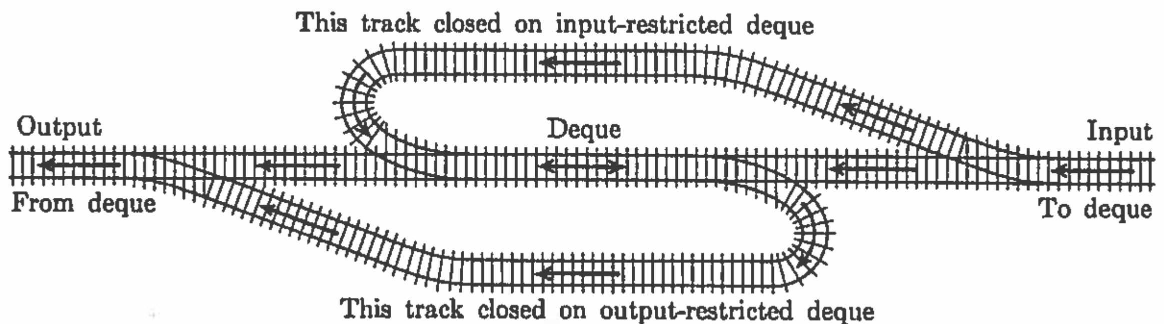



Figure 1. Knuth's Illustration of the Deque as a Railway Switching Network[a]

[a]From The art of computer programming, vol. 1 – 2nd edition (p. 236), by D. Knuth, Menlo Park, CA: Addison Wesley Longman. ©1973 Addisson Wesley Longman Inc. Reprinted with permission.

The advent of computer graphics technology in the late 1970s and early 1980s brought with it new opportunities to illustrate algorithms. Illustrations moved from paper to computer screen, as computer scientists developed the first computer systems to facilitate the creation of so-called *algorithm animations*—animated illustrations of algorithms in action. Due to the limits of the technology of the time, the first system could do little more than aid in the production of algorithm movies (Baecker, 1975). With the help of that system, Baecker produced *Sorting Out Sorting* (Baecker & Sherman, 1981), a legendary instructional film on sorting algorithms (see Figure 2[2]).

---

[2]The grand race depicted here graphically compares nine alternative sorting algorithms operating on an identical 2500-element data set. Data elements to be sorted are represented as scatterplots of dots—one scatterplot for each sorting algorithm. As each respective sorting technique places elements in place, the corresponding scatterplot gradually transforms into an upward-sloping line.

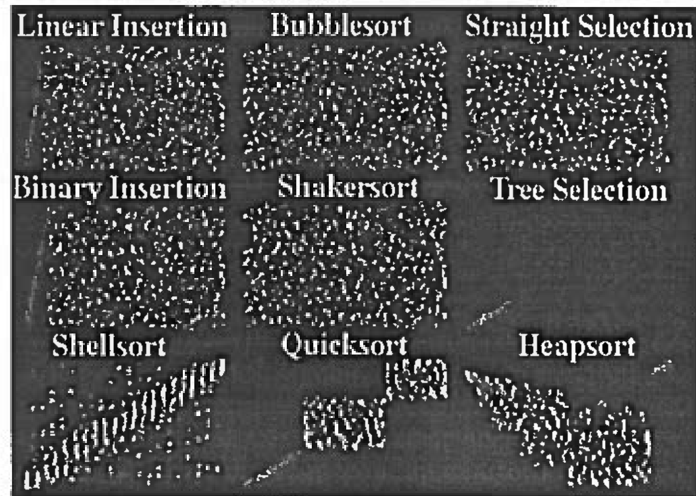



Figure 2. A snapshot of the "Grand Race" in *Sorting Out Sorting*[a]

[a]From Software visualization (p. 379), edited by J. Stasko, J. Domingue, M. Brown, & B. Price, Cambridge, MA: The MIT Press. ©1998 The MIT Press. Reprinted with permission.

By taking advantage of emerging graphical workstation technology, later systems supported *interactive* environments for exploring algorithm pictures and animations. For example, the seminal interactive algorithm animation system, BALSA (Brown, 1988), defined an environment in which users could (a) select sets of input data on which to view an algorithm animation; (b) choose an arrangement of alternative views defined for the animation; (c) start and stop the animation, and control the animation's execution speed; (d) zoom and pan an animation view; and (e) write an animation viewing session to a script for later use. Figure 3[3] presents a snapshot from an interactive session with BALSA.

---

[3]In this snapshot, the user is running a side-by-side comparison of breadth-first and depth-first search executing on an identical graph. The user has chosen to view each algorithm in terms of two identical views. The larger, top views depict the graph being traversed; vertices that have been visited are shaded black, vertices whose descendents have not been fully explored are shaded gray, and vertices that have not yet been visited are shaded white. The animations are presently paused; the user is about to restart them by choosing "Go" from the "Run" menu.

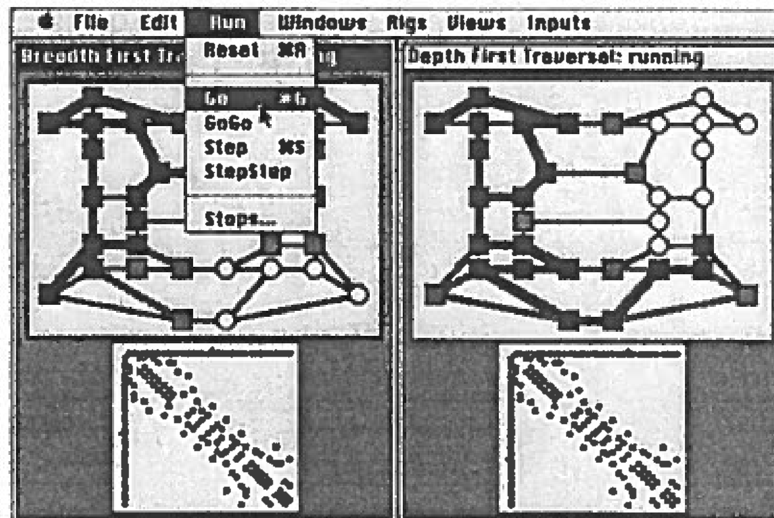



Figure 3. A Snapshot from An Interactive Session with BALSA[a]

[a]From Algorithm animation (p. 65), by M. Brown, Cambridge, MA: The MIT Press. ©1988 The MIT Press. Reprinted with permission.

To date, the most popular application of such systems aims to address the teaching and learning problem described above. So-called *algorithm visualization (AV) systems* enable instructors and students to create and explore graphical representations (both static illustrations and dynamic movies) of the algorithms under study (see, e.g., Brown, 1988; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1989). Noting their increased flexibility as compared with blackboards, whiteboards, and overhead projectors, several computer science instructors have enlisted AV artifacts as visual aids in their lectures (see, e.g., Bazik, Tamassia, Reiss, & van Dam, 1998; Gurka & Citrin, 1996). In addition, some instructors have extolled their value in computer science laboratories (see, e.g., Naps, 1990), as student study aids (see, e.g., Gurka & Citrin, 1996), and as the basis for student assignments (see, e.g., Bazik, Tamassia, Reiss, & van Dam, 1998; Stasko, 1997).

## Problems

Despite the enthusiasm and high expectations of their developers, computer-based AV artifacts have failed to enter mainstream computer science instruction as pedagogical aids (Baecker, 1998; Stasko, 1997). While those few educators who are also AV technology developers tend to employ their own AV technology, the majority of AV educators tend to stick to more traditional pedagogical aids, such as blackboards, whiteboards, and overhead projectors.

In light of the strong intuitive basis for AV artifacts, which is reinforced by the enthusiasm of AV technology developers, a puzzling research question arises:

Research Question: Why has computer-based AV failed to catch on?

While no extant empirical research specifically addresses that question, AV researchers have speculated three main obstacles to AV's widespread adoption:

Obstacle 1: The technology required to deploy AV may not be readily available (Gurka & Citrin, 1996). AV software runs on graphical workstations; college classrooms and laboratories may not be equipped with such workstations, they may not have the right kind of workstations, or they may not have the equipment needed to project an instructor's workstation onto a large screen that an entire class can view.

Obstacle 2: Creating visualizations for classroom use requires substantial time and effort (see, e.g., Brown & Hershberger, 1991; Duisberg, 1987; Helttula, Hyrskykari, & Raiha, 1989; Mukherjea & Stasko, 1994; Stasko, 1989; Stasko, 1991). Mapping an algorithm to an animated representation is a non-trivial problem; it requires careful thought and knowledge of a particular algorithm animation programming framework. Animation

programming is especially difficult if the animation must be *general-purpose*—that is, if it must illustrate the algorithm for all of its possible inputs (Stasko, 1989).

Obstacle 3: The pedagogical value of using AV has not been empirically substantiated (Byrne, Catrambone, & Stasko, 1996; Gurka & Citrin, 1996; Lawrence, 1993; Stasko, Badre, & Lewis, 1993). Recently, AV technology developers have become increasingly interested in subjecting their systems to empirical evaluation. Unfortunately, as several researchers have noted, the results of experiments designed to substantiate the pedagogical benefits of AV technology have been far from convincing (Byrne, Catrambone, & Stasko, 1996; Gurka & Citrin, 1996; Hundhausen, 1997; Kehoe & Stasko, 1996; Stasko, 1997). Indeed, of ten controlled experiments that have aimed to substantiate AV's effectiveness as a pedagogical aid (Byrne, Catrambone, & Stasko, 1996, §2 and 3; Kann, Lindeman, & Heller, 1997; Lawrence, 1993, ch. 4 – 9; Stasko, Badre, & Lewis, 1993), only five have yielded statistically significant results in favor of AV technology. Adding to the indecisive nature of these results is the fact that two of the five significant results could not disentangle the benefits of AV from another factor.

Of course, speculating about these obstacles is all in the interest of addressing a second, closely related research question:

Research Question: Can we develop AV artifacts that overcome these obstacles, allowing AV technology to enter mainstream computer science education?

## Paths to Solutions

Over the past decade, computer science educators, technology developers, and even psychologists have sought solutions to the problem of moving AV technology into the mainstream. Depending both on the obstacle they have chosen to address, and on the primary reason to which they have attributed AV's failure, these researchers fall into one or more of three research camps:

1. Technology Camp: "We need better AV technology." In response to Obstacle 2, the largest research camp has focused on the development of better AV technology (see, e.g., Baecker, 1975; Brown, 1988; Brown & Hershberger, 1991; Citrin & Gurka, 1996; Dominigue, Price, & Eisenstadt, 1992; Duisberg, 1987b; Gloor, 1992; Helttula, Hyrskykari, & Raiha, 1989; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1989). For these researchers, the concept of "better AV technology" has had a rather narrow definition, which stems from a widely held belief that current AV technology does not facilitate the rapid development of visualizations; instructors must invest substantial time and effort in order to create new visualizations with the technology. If creating new visualizations remains a time- and resource-intensive process, they reason, then AV educators are unlikely to adopt the technology. If, on the other hand, they can succeed in building AV technology that facilitates the rapid construction of new visualizations, then the widespread use of AV technology will follow.
2. Pedagogy Camp: "We need better AV-based pedagogy." Responding to Obstacle 3, a second research camp holds that if AV-based pedagogical exercises simply are not effective, then computer science educators are unlikely to adopt AV technology. Accordingly, several computer science educators have explored the

development of improved AV-based pedagogy hand-in-hand with the development of improved AV technology (see, e.g., Brown & Sedgewick, 1984; Eisenstadt, Price, & Dominique, 1993; Kehoe & Stasko, 1996; Knox, 1996; Michail, 1996; Naps, 1990; Sigle, 1990; Stasko, 1997). Lawrence (1993; Lawrence, Badre, & Stasko, 1994), in fact, dedicated an entire dissertation to empirically comparing various pedagogical alternatives with and without AV technology. By providing evidence for and against the adoption of particular AV-based pedagogical approaches, these researchers aim not only to make a case for the value of AV technology as a pedagogical aid, but also to provide guidance for its effective use.

3. Evaluation Camp: "We need better evaluation methods." The third, and smallest, research camp offers a different response to Obstacle 3. This camp stresses the possibility that current AV technology may, indeed, offer real benefits, but that the methods we have employed to measure those benefits have not been sensitive to them (Gurka, 1996; Gurka & Citrin, 1996). Researchers in this camp hold that past failures to produce positive results may be largely a matter of inappropriate or incomplete evaluation methods. Thus, they argue for the need to reexamine critically the results of past empirical studies, with a particular focus on the ways in which these studies have been designed and carried out. By refining current empirical methods, researchers can, this camp believes, put themselves in a position to stage successful empirical studies; guidelines for proper AV technology design and pedagogy, as well as the increased adoption of AV technology, will follow.

## The Thesis

I argue that the problem of underutilized AV technology has its roots in the *theory of effectiveness* that has guided the three camps just described. That theory, which I call Epistemic Fidelity (EF) Theory (after Roschelle, 1990), encompasses commitments to (a) a view of knowledge as representations in people's heads; to (b) a belief in the graphical medium as an effective encoder of knowledge; and, as a consequence, to (c) a view that AV technology is pedagogically effective because it is able to provide a faithful account (i.e., one with high epistemic fidelity) of an expert's mental model of an algorithm, thus enabling the robust and efficient transfer of that mental model to AV viewers. I hold that EF Theory is deficient; thus, progress along any one of the above solution paths effectively perpetuates the theory's deficiencies, leading to, at best, a temporary fix.

The solution, I contend, is to address the problem not at the surface, but at its roots. In other words, instead of tweaking our current design, pedagogy, and evaluation methods, we need to rethink the theory of effectiveness in which they are rooted. Only by proceeding from an alternative theoretical position—one that sheds new light on why AV technology is pedagogically valuable—do we have any hope of overcoming the obstacles that have stood in the way of AV technology's becoming a viable pedagogical aid.

The alternative theoretical position that I explore in this dissertation, *sociocultural constructivism* (see esp. Lave & Wenger, 1991; Wenger, 1998), differs fundamentally from EF Theory. Rather than viewing knowledge and learning at the level of the individual, as EF Theory does, sociocultural constructivism views knowledge and learning at the level of the *community of practice.* On this alternative view, learning is seen not as acquiring target knowledge structures, but rather as participating more centrally in the practices of the community. As I intend to illustrate, this alternative theoretical position has profound

implications for the design, evaluation, and pedagogical use of AV technology. I argue that, by taking the position seriously, we have hope of overcoming the obstacles that have stood in the way of AV technology's adoption.

## Brief Summary of the Dissertation

Clearly, one cannot hope to furnish a convincing empirical case for an alternative theory within the scope of a single dissertation. Accordingly, the more modest objective of this dissertation is to lay a preliminary foundation for the thesis, on which future research can build. The preliminary foundation I am striving for should both establish the plausibility of the alternative theoretical position, and generate excitement about and interest in its further development. I aim to establish the plausibility of the position by furnishing and analyzing a corpus of empirical data; I aim to generate excitement and interest by presenting a novel prototype AV system that differs radically from the systems that have been built so far.

Figure 4 presents a graphical summary of the dissertation. The labels (a) through (f) each map to a major component of the dissertation. In the subsections that follow, I present an executive summary of the dissertation by elaborating on these components in turn.

### EF Theory, Its Stronghold on Past Research, and Its Inadequacy

My first objective is to describe EF Theory [abbreviated "EF" in Figure 4(a)] and to demonstrate its stronghold on past AV research through a review of the literature. By performing an informal meta-analysis of the experimental research whose results are summarized in my discussion of Obstacle 3 above, I make the case that EF Theory is

inappropriate as a guiding theory of effectiveness—hence the slash through the EF Theory oval in Figure 4(a).

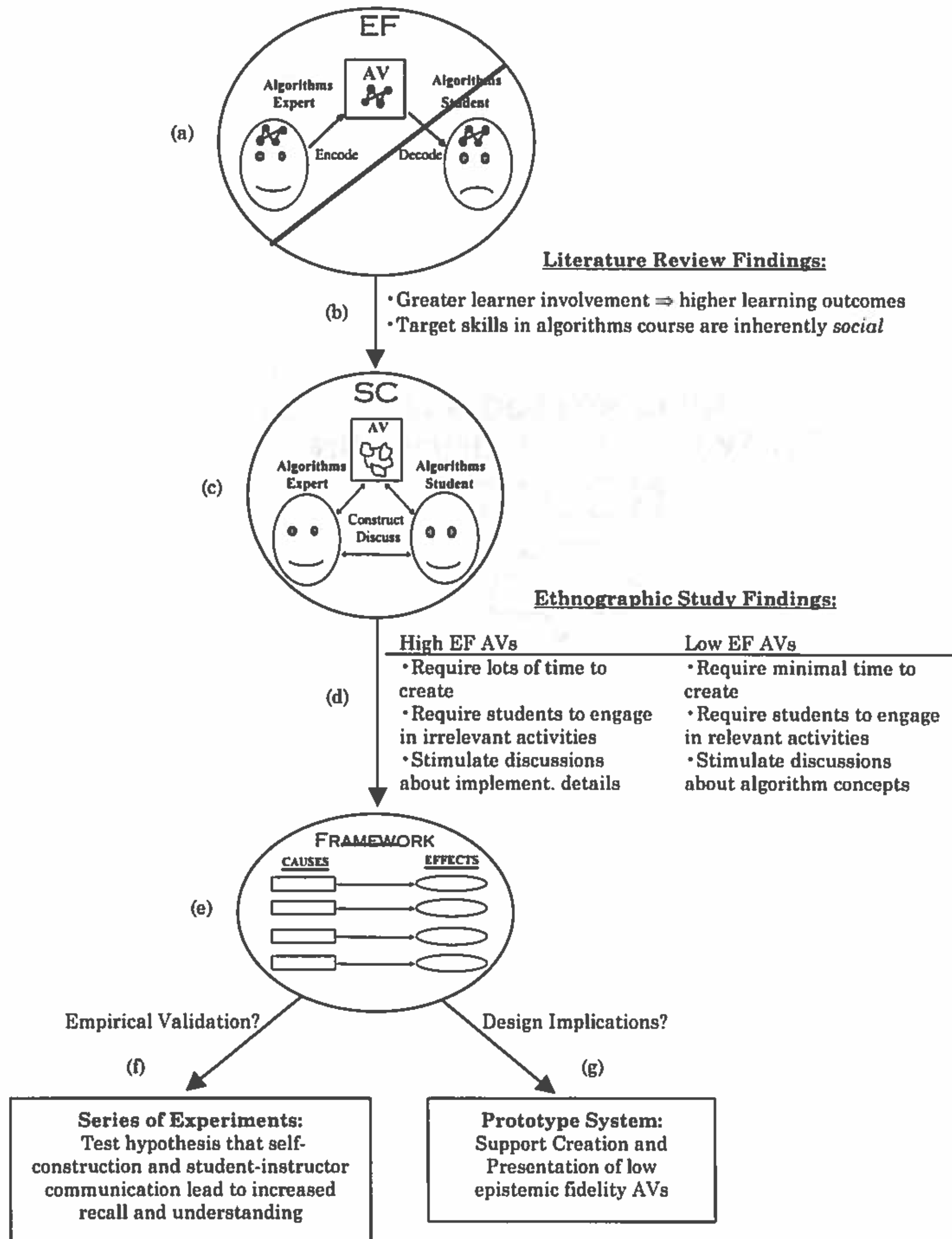



Figure 4. Graphical Summary of the Dissertation

## Choosing an Alternative Theory

My second objective [see Figure 4(b)] is to choose a more promising alternative theoretical foundation from which to proceed. In addition to arguing against the validity of EF Theory, my meta-analysis of past experimental research reveals a noteworthy trend: namely, that the more *actively* learners were involved with AV technology, the better they performed. Students who were actively involved did such things as (a) develop their own input data sets and observe the execution of the AV on those data sets; (b) program the algorithm while observing the AV; and (c) make explicit predictions regarding future states of the AV before seeing them. This finding, coupled with the observation that the target skills of an algorithms course are inherently social, argues for a shift to a theoretical position called *sociocultural constructivism* (see, e.g., Lave & Wenger, 1991). Sociocultural constructivism [abbreviated "SC" in Figure 4(c)] views knowledge not at the level of the individual, but at the level of the community of practice. On this alternative view, learning is seen in terms of participating more centrally in the practices of the community; AV technology is regarded as pedagogically valuable insofar as it facilitates access to increasingly "expert" community practices, and insofar as it mediates meaningful conversations among community members.

## Empirically Exploring the Sociocultural Constructivist Approach

My third objective is to adapt the sociocultural constructivist position so that it accounts for the pedagogical role of AV technology within the community of practice being reproduced through an undergraduate algorithms course. To do so, I present a pair of ethnographic studies of an undergraduate algorithms course in which AV software was used

as a means of facilitating students' more central participation in the community. Specifically, students were asked to use both AV software, and simple art supplies, to construct and present their own visualizations—two activities commonly performed only by community experts (algorithms instructors).

The two key findings of these studies [see Figure 4(d)] not only support the sociocultural constructivist position, but also offer valuable practical recommendations for how best to implement AV construction assignments in an undergraduate algorithms course. First, the studies found that assignments in which students were required to use conventional AV software to construct *high epistemic fidelity* AVs actually distracted them from the focus of the course. Here, the term "high epistemic fidelity" denotes AVs that (a) are capable of depicting the target algorithm for general input, and (b) resemble the polished figures found in algorithms textbooks. Constructing high epistemic fidelity AVs was distracting not only because it took large amounts of time (over 30 hours on average, according to data I collected), but also because it required students to engage in low level implementation activities—plainly not the focus of a course that stresses the conceptual foundations of algorithms. On the other hand, when students used simple art supplies to construct low epistemic fidelity AVs, they spent far less time overall, and the time that they did spend was devoted to more relevant activities. Thus, low epistemic fidelity technology enabled students to participate more extensively in community-relevant practices—an important condition for learning, on the sociocultural constructivist view.

The second key finding of the studies was that, when students presented their AVs and discussed them with their instructor and peers, high epistemic fidelity AVs tended to generate conversations about implementation details, whereas low epistemic fidelity AVs tended to generate more relevant conversations about algorithm concepts. It appears that high epistemic fidelity AVs were so difficult to create that students preferred to discuss the

toil they put into implementing them. On the other hand, the rough, unfinished look of low epistemic fidelity AVs seemed to invite conversations about the concepts they represented. Once again, this finding resonates with the sociocultural constructivist view, which regards the presentation and discussion of algorithms as a key form of participation in an algorithms course, and which sees AV technology's value in terms of its ability to facilitate such participation.

## A Framework of Cause and Effect

My fourth objective is to use these findings as a basis for developing a set of specific hypotheses regarding the effectiveness of AV technology in an undergraduate algorithms course. The result is a framework of cause and effect [see Figure 4(e)]. In my ethnographic studies, I identified six causal factors as strongly influencing the effectiveness of AV technology, and four distinct measures emerged as appropriate gauges of the effectiveness of these factors. In addition to elaborating on these factors and measures, the framework proposes four specific hypotheses, each of which links one or more causal factors to a specific dependent measure:

1. *The Activity Relevance Hypothesis*: Low input generality, low typeset fidelity, and direct graphics cause high activity relevance
2. *The Communication Effectiveness Hypothesis:* Low epistemic fidelity causes high communication effectiveness
3. *The Understanding and Recall Hypothesis*: Self-constructing AVs with a story line, and then presenting them to an instructor for feedback and discussion, causes high recall and understanding

4. *The Community-Building Hypothesis*: Self-construction and high instructor communication cause high community-building

The framework constitutes the beginnings of an alternative theoretical position that the remainder of the dissertation begins to explore and flesh out, and that future research will need to develop and refine further. Resonant with constructivist learning theory, the position stresses (a) the value of students' constructing and refining their own personal representations of the material they are learning; and (b) the importance of students' participating more centrally in algorithms practice by presenting their self-constructed AVs to their peers and instructor for feedback and discussion. Moreover, in radical departure from extant AV technology's obsession with high epistemic fidelity AVs, the position underscores the value of low epistemic fidelity AVs in focusing students' on algorithms and how they work, and in promoting effective communication about algorithms.

## Two Research Directions

Aside from refocusing the ongoing dialog on AV effectiveness, the hypotheses posited by the framework raise numerous research questions that are grist for further investigation. The final objective of the dissertation is to present original research that addresses two of the most important of these questions. The first research question [see Figure 4(f)] concerns the viability of the hypotheses: Can they be empirically validated? To begin to answer this question, I present an experiment that puts to the test a key piece of the Understanding and Recall Hypothesis (see above), whose confirmation should prove to be of great interest to computer science educators:

Hypothesis: On a test of procedural understanding and recall, students who construct their own AVs will outperform students who view and interact with an AV constructed by an expert.

Notice that, while EF Theory would not predict this hypothesis, the hypothesis follows directly from constructivist theory.

To test this hypothesis, the experiment employed a between-subjects design with two treatment groups:

| | |
|---|---|
| *Self-Construction*: | Students use simple art supplies to construct their own homemade visualization of an algorithm. They are responsible for using their visualization to step through the execution of the algorithm for at least five input data sets. |
| *Active Viewing*: | Students interact with a visualization of predefined visualization of the same algorithm constructed by an expert. They are responsible for watching the visualization execute on at least five input data sets of their choosing. |

For this experiment, I recruited 24 students out of a later offering of the same undergraduate algorithms course in which I conducted ethnographic fieldwork. Participants were randomly assigned to one of the two treatments such that the two treatment groups were optimally matched with respect to self-reported grade point average. Participants had two and a half hours either to construct their own visualization (Self Construction group), or to interact with a predefined visualization (Active Viewing group), of QuickSelect, a divide-and-conquer algorithm that had been covered in a course lecture prior to the study (see, e.g., Cormen, Leiserson, & Rivest, 1990, pp. 153–155, 185, 187–188). Upon completing the learning phase of the study, participants completed two tasks designed to assess their procedural understanding and their recall ability:

1. A tracing task in which they had to trace through the algorithm for a novel set of input data; and
2. a programming task in which they filled in a partially-completed C++ or Java implementation of the QuickSelect algorithm.

Participants had 25 minutes to complete each of four tracing tasks, and 35 minutes to complete the programming task. I scored participants' traces and programs by comparing them against perfect traces and programs.

Although the Self Construction group outperformed the Active Viewing group on both tasks, statistical analyses failed to yield any significant differences between the two groups. A thorough review of the experimental materials and procedure did not reveal any flaws in the design; participants appear to have had an excellent chance of doing well on all tasks. This is borne out in the data, which, although extremely variable, include seven scores above 80% on both the tracing and programming tasks. In the absence of experimental design flaws, the most plausible reason that no significant differences were detected is that the self-construction factor, as manipulated in this experiment, is simply not reliable enough to produce an effect.

On a practical level, however, the results of this experiment can be placed in a positive light. Using conventional (high epistemic fidelity) AV technology to prepare visualizations for classroom use is notoriously difficult (see, e.g., Brown & Sedgewick, 1985). In contrast, having students construct their own low epistemic fidelity visualizations using art supplies is cheap, easy, and requires a relatively small investment of time on the part of instructors. Thus, instructors looking for a low-overhead way to incorporate AV technology into their curricula can take comfort in the finding that students who use "low tech" art supplies to construct their own visualizations may learn algorithms just as well as students who interact with AVs developed by their instructor with conventional "high tech" AV technology.

The second research question explored by this dissertation addresses the implications of the framework's hypotheses for technology design [see Figure 4(g)]: If one takes the hypotheses as constraints on the design space of effective AV technology, what kind

of AV system emerges? I pursue this question by designing and prototyping (a) SALSA (Spatial Algorithmic Language for StoryboArding), a high-level, interpreted language for programming low epistemic fidelity AVs; and (b) ALVIS (the ALgorithm VIsualization Storyboarder), an interactive graphical user interface for programming in SALSA. In line with the framework's hypotheses, SALSA and ALVIS aim to meet two key functional requirements:

1. Support the quick and easy construction of *low epistemic fidelity* AVs; and
2. Support the interactive presentation and discussion of those AVs.

Notice that these requirements differ markedly from those that follow from EF Theory, which has prompted extant AV systems to support the creation, but not the presentation, of *high epistemic fidelity* AVs.

To meet the first of the above requirements, SALSA and ALVIS aim to make constructing a low epistemic fidelity animation (i.e., a "storyboard") as easy as, and ideally easier than, constructing a homemade animation out of simple art supplies. To do so, ALVIS and SALSA are firmly rooted in the physical metaphor of art supply storyboard construction. Indeed, in past empirical studies (Chaabouni, 1996; Douglas, Hundhausen, & McKeown, 1995, 1996), and in my ethnographic fieldwork, creating homemade visualizations out of simple art supplies proved incredibly natural, quick, and easy.

An important component of this metaphor is the concept of *cutouts* (or *patches*; see van de Kant, Wilson, Bekker, Johnson, & Johnson, 1998): pieces of construction paper that may be cut out and drawn on, just like real construction paper. SALSA/ALVIS users create homemade animations by cutting out, sketching on, and arranging cutouts on a static background, and then specifying the ways in which the cutouts should be animated over time.

The ALVIS interactive environment includes both a “SALSA script” window, which contains the SALSA code that drives a homemade animation, and a “Storyboard” window in which the animation actually unfolds. Users can create, place, and animate cutouts either by typing SALSA commands directly into the “SALSA script” window, or by directly manipulating elements in the “Storyboard” window (which, in turn, automatically generates and inserts SALSA commands into the “SALSA script” window).

Observe that specifying an AV with ALVIS and SALSA differs markedly from specifying an AV with conventional AV technology. Whereas conventional AV technology (see Roman & Cox, 1993 for an overview) requires one to specify explicit mappings between an underlying driver program and an animation, ALVIS and SALSA enable one to specify animations that drive themselves; the notion of an external driver program is jettisoned altogether. To support self-driving animations, SALSA enables the logic of an animation to be specified in terms of the animation's *spatiality*—that is, in terms of the *spatial relations* (e.g., *above*, *left of*, *between*) among cutouts in the animation.

The following brief example illustrates the kinds of spatial logic that SALSA supports. Recall that the idea behind the *insertion sort* algorithm is to successively insert elements in the unsorted portion of the array into the sorted portion of the array. The pseudocode presented in Figure 5 describes the algorithm.

One possible animation of the algorithm represents sorting elements as blocks, and uses a special arrow to mark the dividing line between the sorted and unsorted portions of the list (Figure 6). Each time the outer loop in the insertion sort is executed (lines 2 to 11), the arrow moves one position to the right.

```
 1:  INSERTION-SORT(A)
 2:  for xindex ← 2 to n - 1 do
 3:    x ← A[xindex];
 4:    j ← xindex - 1;
 5:    while (j > 0) and (A[j] > x) do
 6:      A[j+1] ← A[j];
 7:      j ← j - 1;
 8:    end while;
 9:    A[j+1] ← x;
11:  end for;
12:  end INSERTION-SORT;
```

Figure 5. Pseudocode Description of Insertion Sort Algorithm

Notice that the internal logic governing the insertion sort *algorithm* is distinct from, but parallel to, the internal logic governing the insertion sort *animation*. For instance, consider the way in which each decides when the array is sorted. In the insertion sort algorithm, we continue sorting until the outer loop index (*xindex*) reaches $n - 1$—that is, until it is one less than the size of the array *a*. By contrast, in the insertion sort animation, we continue with the animation until the arrow slides to the right of the row of blocks. Whereas in the algorithm, the internal logic is mathematical ($xindex = n - 1$), in the animation, the internal logic is spatial (*arrow is right of edge of row of blocks*). This is precisely the kind of spatial logic, which might be thought of as analogy between the algorithm and the animation, that SALSA is capable of expressing.

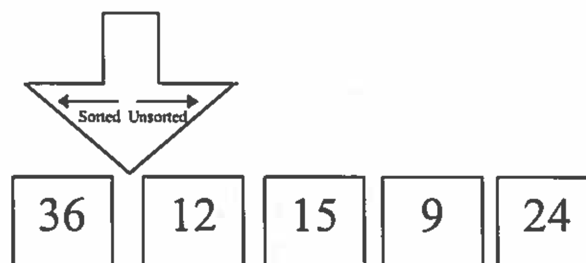



Figure 6. An AV of the Insertion Sort Algorithm

Recall that the second requirement of SALSA and ALVIS is that they must support the interactive presentation and discussion of low epistemic fidelity AVs. To meet this requirement, ALVIS supports the following two interface features:

1. *Fine-grained execution control.* Users may execute or single-step through an animation both forwards and backwards. At any point, users may pause, change the execution speed, or reverse the direction of execution.
2. *Interactive mark-up and modification.* Users may annotate the animation as it is executing with a "marking pen" tool; annotations may be erased at any point with an "eraser" tool. In addition, users may pause the animation and modify (i.e., reprogram) it at any point in its execution—without having to restart the animation from the beginning.

As was the case for animation specification, it is important to underscore that ALVIS and SALSA support a range of functionality for animation presentation and discussion that differs markedly from that of extant AV technology. To appreciate the contrast, consider the user interface that POLKA (Stasko & Kraemer, 1993) provides for controlling the execution of animations (see Figure 7); this interface is typical of extant AV systems. Much like the interface of a tape recorder, it allows users to start and pause the animation at any point, and to adjust the execution speed. Moreover, in POLKA, along with most extant AV systems, making changes to an animation requires one to quit the animation, modify code (which is typically low-level), recompile the animation, and restart the animation. Clearly, within the scope of an interactive presentation, performing this list of activities would be at best distracting, and at worst downright impossible.

Figure 7. The POLKA User Interface for Controlling Animation Execution

## Organization of the Dissertation

The dissertation summarized in the previous section is presented in the seven chapters that follow. Chapter II introduces EF Theory, reviews its influence on past research, and performs a meta-analysis of the experimental research designed to support it. That meta-analysis motivates the shift to Constructivist learning theory. Chapter III presents the cognitive and the sociocultural versions of Constructivism, and argues that, because the target skills of an algorithms course are inherently social, the sociocultural version is more appropriate as a guiding theory. Chapter IV presents two ethnographic studies that explore the potential of the sociocultural constructivist pedagogical approach. Based on observations made in the ethnographic studies, Chapter V develops a framework of cause and effect—a specific set of hypotheses regarding the effectiveness of AV technology within an undergraduate algorithms course. Chapter VI proposes a series of experiments for testing one of those hypotheses, and presents the results of the first experiment in the series. Chapter VII describes SALSA and ALVIS, a prototype language and system that explore the design implications of the hypotheses. Chapter VIII summarizes the work and its contributions, and outlines areas for future research.

# CHAPTER II

# EPISTEMIC FIDELITY THEORY

> What makes a[n algorithm] visualization work in a lecture. . .is a short cognitive distance between concept and visualization. How closely the graphics on the screen correspond to the mental model is the best measure of a visualization's usefulness in class. When the mapping is direct and obvious, the representation is transparent and students "get it" right away. They can then concentrate on the ideas being illustrated, rather than the illustration itself.
>
> (Bazik, Tamassia, Reiss, & van Dam, 1998, p. 386)

A central argument of this thesis is that the failure of AV technology to enter mainstream CS education is rooted in fundamental deficiencies in a particular theoretical framework that has guided past research. This chapter lays the groundwork for the rest of the dissertation by (a) precisely identifying the particular theoretical framework at issue, (b) linking it to past research into AV technology design, evaluation, and pedagogy, and (c) presenting the empirical and pragmatic case against it.

The body of research into AV technology design, evaluation, and pedagogy has grown significantly over the past fifteen years. Price, Baecker, and Small (1993) estimate that papers on over 100 software visualization systems have been published. At least ten controlled experiments (Byrne, Catrambone, & Stasko, 1996, §2 & 3; Kann, Lindeman, & Heller, 1997; Lawrence, 1993, ch. 4–9; Stasko, Badre, & Lewis, 1993) explicitly evaluating AV technology's effectiveness have been reported. Finally, the literature contains the experience reports of at least a dozen computer science educators who have actually used the technology in their curricula (see, e.g., Bazik, Tamassia, Reiss, & van Dam, 1998; Brown, 1988, Appendix A; Brown & Sedgewick, 1984; Cox & Roman, 1994; Eisenstadt, Price, & Dominique, 1993; Gurka & Citrin, 1996).

On the surface, the *guiding principles* of the three lines of AV research appear to have little in common. Classic software engineering methodology, which holds modularity and reuse inviolate, has been the main guiding force behind AV systems research. Classic experimental psychology, with its interest in imposing tight environmental controls in order to ensure generalizability and replicability, has heavily influenced AV systems evaluation. The traditional didactic teaching practices of the Academy have shaped the use of AV technology in the classroom.

Yet, if one delves below the surface, I suggest that one finds a unified theoretical framework underlying the bulk of this seemingly heterogeneous research. That framework, which I call *Epistemic Fidelity* Theory[4], articulates a particular view of how and why AV technology is effective. The foundation of the EF view is a set of assumptions about what knowledge is, how it is acquired, and the efficacy of graphical representations in that acquisition. Below, I introduce the Allegory of the Musica as a means of highlighting the key assumptions of the theory. Through a critical review of the literature, I then illustrate the influence of the EF view on the research into AV design, evaluation, and pedagogy. Finally, I show that past empirical research into AV effectiveness fails to substantiate EF Theory's predictions. The case against EF Theory presented here motivates the need for an alternative guiding theory, which I introduce in Chapter III.

## The Allegory of Musica

Roaming the faraway land of Musica, the Musibeest (see Figure 8) was a large land mammal endowed with the ability to produce beautiful songs. Legend has it that ancient

---

[4]I borrow this term from Roschelle (1990), who, in turn, borrows it from Wenger (1987); see pp. 312–314.

Musican civilizations held the Musibeest's songs in such esteem that they attempted to domesticate the animals. Their hope was to train them to sing the songs on demand.

Figure 8. The Musibeest

Unfortunately, over a millenium, the domestication process gradually killed off all but a precious few Musibeests. The near extinction of the Musibeest compelled the government to protect them. The Musican government thus established the Department of the Musibeest, an exclusive government agency charged with the mission of taking into captivity, and caring for, all remaining Musibeests.

The Musicans were a singing people, and all Musicans were endowed with a unique ability to sing the songs of the Musibeest. In fact, in Musica, the ability to reproduce Musibeest songs was regarded as the highest form of knowledge. Since so few Musibeests were around, Musicans had to rely on carefully-prepared recordings to learn the songs of the Musibeest. The Musicans developed a schooling system centered around such recordings, which, because of the general inaccessibility of Musibeests, only Musibeest instructors were allowed to make. Depending on a given instructor's knowledge of the Musibeest song, the recordings she created, and subsequently used in her instruction, more or less reflected the true nature of the Musibeest songs.

To teach students the songs of the Musibeest, instructors would rely extensively on lectures based on their recordings. In addition, instructors would often give students copies of the recordings to explore on their own. Students would take these recordings home and

play them on their personal stereo systems. By repetitively listening to the recordings—both in lectures, and on their own—students would gradually internalize the songs of the Musibeest, allowing them to reproduce the songs with stunning accuracy.

The capstone of the Musican schooling system was a final exam. For the exam, instructors would first record students' own renditions of the songs, and then send the recordings to the Department of the Musibeest. Members of that agency would use a special device to compare systematically each student's voice against that of an actual Musibeest. Based on the closeness of the match, each student received a grade. Those who could produce highly faithful accounts of Musibeest songs achieved honorable status in Musica, and were invited to become Musibeest instructors. The best Musibeest instructors, as indicated by the accuracy with which their students could mimic Musibeest songs, might be invited to join the Department of the Musibeest. Such an invitation was the highest honor a Musican could hope for.

## Key Assumptions

What does the Allegory of the Musibeest have to do with AV effectiveness? Below I present four key assumptions highlighted by the allegory: (a) The Knowledge Representation Assumption, (b) The Knowledge Flow Assumption, (c) The Graphical Medium Effectiveness Assumption, and (d) The Epistemic Fidelity Assumption. These assumptions form the foundation of EF Theory. In preparation for my critical review of the AV technology, evaluation, and pedagogy literature, I conclude by sketching out the practical implications of these assumptions—that is, just what it means for an AV to have high epistemic fidelity.

## The Knowledge Representation Assumption

For the Musicans, knowledge is the song of the Musibeest; it exists independently of humans, but can be instantiated by humans who sing the songs. Analogously, according to the first key assumption of EF Theory is that knowledge exists independently of humans, but can be instantiated as symbolic structures in humans' heads. This assumption has its roots in an epistemological framework called Representationalism (see, e.g., Newell, 1980), as Doerry (1995) notes:

> The central tenet of Representationalism is that we carry inside our heads symbolic models, or *representations*, of the physical world and its behavior as well as of our intentions, goals, and beliefs with respect to the world, and the actions we can perform (i.e. plans) to achieve certain goals; these symbolic models serve as the basis for all reasoning and action that we perform. (p. 24)

## The Knowledge Flow Assumption

In the land of Musica, people gain understanding—that is, an ability to reproduce Musibeest songs—by listening to high-fidelity recordings of the Musibeest. Thus, in Musica, target knowledge is encoded by a high-fidelity recording, and decoded by a listener; it essentially flows from Musibeest to Musican by way of a Musibeest instructor's recording. Analogously, EF Theory holds that an algorithm expert's knowledge is encoded in an AV, and decoded by an AV viewer. Thus, knowledge is seen to flow from expert to AV to viewer through the "conduit" (Reddy, 1977) of the visual medium.

Moreover, the encoding and decoding process of the transmission is seen to be *deterministic*; there exists a "deterministic correspondence between symbolic forms and their significance" (Doerry, 1995; see also Wenger, 1987). Consequently, a viewer's failure to obtain the knowledge structures encoded by an AV cannot be due to a misinterpretation of

the signals; rather, it can only be attributed to "an error in transfer. .caused by some flaw or inadequacy in the medium" (Doerry, 1995).

## The Graphical Medium Effectiveness Assumption

The Musicans believe that high-fidelity audio tape is endowed with an excellent ability to capture the songs of the Musibeest. Analogously, the EF view holds that graphical representations (such as AVs) are endowed with an excellent ability to support representations that closely match an expert's mental model of an algorithm—that is, the way in which an expert conceives of the algorithm and how it works. The belief is that graphical representations peel away unnecessary detail, presenting an expert's mental model at a level of abstraction that is "just right." The resulting match, it is held, is generally closer than that which can be achieved with other media.

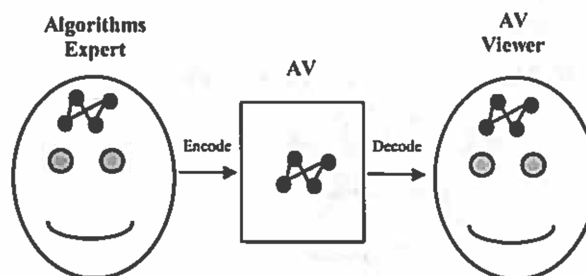



Figure 9. A Schematic Diagram of Knowledge Flow According to EF Theory

## Overriding Assumption: Epistemic Fidelity Is Crucial

In Musican education, the more faithfully an instructor's recording captures the songs of the Musibeest, the more accurately her students will be able to reproduce Musibeest

songs. Note that this conclusion logically follows from the previous three assumptions. Analogously, the overriding assumption of the EF view asserts that a close denotational match—that is, one with *high epistemic fidelity*—between an algorithm expert's knowledge structures and an AV leads to robust, efficient acquisition of those structures by the viewer, who non-problematically decodes and internalizes the target knowledge. This assumption is graphically depicted in Figure 9.

## Practical Implications

In practical terms, what might it mean for an AV to support a high epistemic fidelity match with an expert's mental model—that is, the way in which an expert thinks about the algorithm and how it works? AV technologists emphasize that the value of a computer-based AV lies in its ability to depict *faithfully the execution of the underlying algorithm*. The assumption, as articulated by Baecker (1998), is that an AV that is "constructed automatically as a by-product of [an algorithm's] execution. .[is] guaranteed to portray this execution faithfully" (p. 369).

However, this assumption seems to overlook the fact that AVs exist as a result of human intervention. That AVs are not simply structures of logic—that they do indeed require human intervention in order to gain the putative faithfulness with which they are endowed—follows from Brown's (1988) explanation for why AVs cannot be generated automatically:

> Algorithm animation displays cannot be created automatically because they are essentially monitors of the algorithm's fundamental *operations*; an algorithm's operations cannot be deduced from an arbitrary algorithm automatically but must be denoted by *a person with knowledge of the operations performed by the algorithm*
> (p. 18, italics added)

As a member of a particular community that is interested in algorithms, the person to whom Brown refers has a sense both of what is important about algorithms, and of how to represent what is important in a way that others will understand.

We see, then, that EF Theory actually implies two distinct senses in which an AV derives epistemic fidelity:

1. *The algorithmic sense.* An AV with high epistemic fidelity maintains the intended correspondence with the dynamic behavior of the algorithm. In other words, at each point in its execution, the AV maps back to exactly that valid state in the underlying program which the AV author intends to portray. Conversely, for each state in the underlying program that is of interest to the AV author, there exists a state in the AV that depicts the interesting algorithm state as the AV author intends.
2. *The cultural sense.* An AV with high epistemic fidelity portrays the underlying algorithm in terms of the abstractions, events, and properties that expert algorithmaticians deem important and noteworthy. Furthermore, in order to represent those abstractions, events, and properties, it makes use of the notation and conventions established by expert algorithmaticans.

In short, an AV with high epistemic fidelity not only encodes the "correct" mental model of the underlying algorithm, but it does so using language that is meaningful to its viewers.

## Variations on EF Theory

The version of EF Theory articulated above can be considered *strong* in the sense that it asserts that knowledge transfer occurs without regard to any other factor except epistemic fidelity. It is important to note that not all AV research subscribes to the strong

version of EF Theory. Especially in the Pedagogy and Evaluation Camps, weaker versions of the theory have been explored. While the same underlying epistemology is firmly intact in these weaker versions, they hold that certain characteristics above and beyond epistemic fidelity play an important role in successful knowledge transmission. Below, I describe three different weak versions that have been considered.

## Weak EF Theory (Learner Involvement)

Recent empirical research (Byrne, Catrambone, & Stasko, 1996; Lawrence, 1993, ch. 6 and 9) has led to the development of a weak version of EF Theory in which the AV viewer's level of attention, in addition to epistemic fidelity, plays an important role in successful knowledge transmission. Specifically, this weak version of EF Theory differs from the strong version in that it asserts that the decoding process requires the AV viewer's attention to proceed without error; the more heightened the AV viewer's attention , the more robust and efficient is the AV viewer's decoding of the AV.

Furthermore, according to weak EF Theory, heightened attention can be fostered by increasing the learner's *level of involvement*. To be more involved, a learner might do such things as construct her own data sets (Lawrence, 1993, ch. 6, 9) make explicit predictions regarding the next steps of the AV (Byrne, Catrambone, & Stasko, 1996), program an algorithm while viewing the AV (Kann, Lindeman, & Heller, 1997), or even construct her own AV (Kann, Lindeman, & Heller, 1997).

## Weak EF Theory (Individual Differences)

A second variant of weak EF Theory ascribes significance to *individual differences.* Lawrence (1993) and Gurka and Citrin (1996) hypothesize that measurable differences in human spatial and cognitive abilities enable some individuals to decode AVs more efficiently and robustly than others. The greater the AV viewer's spatial and cognitive abilities, the more robust and efficient is the AV viewer's decoding of the target knowledge.

## Weak EF Theory (Dual-Coding)

Based on Mayer and Anderson's (1991) *integrated dual-code hypothesis*, a third version of weak EF Theory proceeds from Paivio's (1983) assumption that "cognition consists largely of the activity of two partly interconnected but functionally independent and distinct symbolic systems" (p. 308). One encodes verbal events (words); the other encodes nonverbal events (pictures). According to Mayer and Anderson's hypothesis, AVs that encode knowledge in both verbal and non-verbal modes allow viewers to build dual *representations* in the brain, and *referential connections* between those representations. As a consequence, such AVs facilitate the transfer of target knowledge more efficiently and robustly than AVs that do not employ dual-encoding.

## EF Theory's Influence on Past Research

In this section, I critically review past research into the design, evaluation, and pedagogical use of AV technology. My goal is to make the case that this body of research has pursued research agendas that have been quietly shaped by EF Theory's assumptions. In

AV systems research, the strong version of the theory has been the primary guiding force; in evaluations of AV effectiveness, and in discussions of pedagogical applications, various weak versions have gained prominence. Note that whenever I refer to EF Theory below, I shall always mean the strong version of EF unless I explicitly state otherwise.

## Influence on AV Technology Design

In Western culture, pictures in general, and animated computer graphics in particular, have widespread intuitive appeal as effective media for communication. The Graphical Medium Effectiveness Assumption (see p. 28) codifies this intuitive appeal, and was clearly a powerful motivating force behind the pioneering research into AV technology. Consider, for example, Baecker's reflections on *Sorting Out Sorting* (Baecker, 1981), the thirty minute film on sorting algorithms that is widely regarded as the seminal research of the modern AV technology era:

> The film goes beyond a step-by-step presentation of the algorithms, communicating an understanding of them as dynamic processes. We can see the programs in process, running, and we therefore see the algorithms in new and unexpected ways. We see sorting waves ripple through the data. We see data reorganize itself as if it had a life of its own. These views produce new understandings which are difficult to express in words. (Baecker, 1998, p. 377)

The advent of graphical workstations in the early 1980s, shortly after the release of *Sorting Out Sorting*, provided technologists with a medium far more interactive than film. In his seminal dissertation on graphical workstation-based AV technology, Brown (1988) is clearly as inspired as Baecker by the apparent effectiveness of computer graphics as a communication medium. Indeed, in the dissertation's opening chapter, Brown heralds the power of the graphical medium in promoting a new and improved understanding of algorithms:

> An algorithm animation environment is an "exploratorium" for investigating the dynamic behavior of programs, one that makes possible a fundamental improvement in the way we understand and think about them. It presents multiple graphical displays of an algorithm in action, exposing properties of the program that might otherwise be difficult to understand or might even remain unnoticed (p. 1).

Though they avoid explicit mention of knowledge representation, one might infer from quotes like the ones above that AV technologists believe knowledge to be represented by AVs. How else, one must ask, could "views produce new understandings" or "expos[e] properties of [a] program that might otherwise be difficult to understand?"

AV technology's loyalty to the remaining assumptions of EF Theory is evidenced by its direct support of specific features and techniques pioneered in Brown's (Brown, 1988; Brown & Sedgewick, 1984; Brown & Sedgewick, 1985) seminal research, and widely-embraced ever since (see, e.g., Duisberg, 1987b; Helttula, Hyrskykari, & Raiha, 1989; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1989). I detail these EF-supporting features and techniques in the following three subsections.

## Support for Knowledge Flow

AV technologists' loyalty to the Knowledge Flow Assumption (see p. 27) manifests itself in at least two forms: (a) their interest in supporting increasingly high-bandwidth output media; and (b) the dyadic user model on which AV artifacts are based.

*Support for increasingly high-bandwidth output.* Advances in computational power over the past decade have set the stage for the development of increasingly sophisticated output technology, including color, sound, and three-dimensional (3D) graphics. AV technologists have been eager to incorporate such technology into AV systems, and to develop techniques for its use. For example, Brown and Hershberger (1992) develop techniques for incorporating color and sound into AVs. Likewise, Lieberman (1989), Roman et al. (1992),

and Stasko and Wehrli (1993) present AVs that make use of the third dimension. Brown and Najork (1993) present a collection of techniques for using the third dimension in AVs.

The way in which AV technologists motivate their exploration of these new output technologies indicates a loyalty to the Knowledge Flow Assumption. In their work on three-dimensional AV, Brown and Najork (1993) sum up the motivation behind the AV Technology Camp's interest in color, sound, and 3D as follows: "The third dimension provides *an extra degree of freedom for conveying information*, much as color adds to black-and-white images, animation adds to static images, and sound adds to silent animations." (p. iv, italics added). As the quote suggests, AV technologists regard advances in output technology as opportunities to explore techniques for encoding more knowledge into AVs than was previously possible. From the strong version of EF Theory, it follows that AVs that encode more knowledge necessarily lead to the transfer of more knowledge to the AV viewer. Similarly, the *dual-coding* version of EF Theory asserts that information that is dually-encoded in both pictures and sounds, or pictures and words, leads to more robust knowledge transfer. Whether AV technologists involved in this movement were guided by strong EF or by the dual-coding version of weak EF, the influence of the Knowledge Flow Assumption on AV technolgists is clear: AV technologists are interested in exploiting color, sound, and 3D precisely because they believe those technologies will effectively increase or enhance an AV's ability to transfer knowledge.

*Dyadic user model.* In the opening chapter of his dissertation, Brown (1988) introduces a dyadic conceptual model for AV technology based on two conceptual actors: *client programmers*, who map algorithms to AVs; and *end-users*, who view and interact with those AVs.

According to this model, an AV system provides an *algorithm animation system* for client programmers, and an *end-user environment* for end-users. These two interfaces are

not only conceptually distinct, but technologically distinct as well, as Brown (1988) notes: "The algorithm animation system is the code with which client-programmers interface, and the algorithm animation environment is the runtime environment that end-users see. It is the result of compiling the code that client-programmers implement with the algorithm animation system" (pp. 6–7).

Recall that, according to EF Theory, an algorithm expert encodes expert knowledge in an AV, and an AV viewer decodes that knowledge. Figure 10 illustrates the obvious conduits for knowledge encryption and decryption built into Brown's conceptual model. An AV artifact's programmer interface can be seen as a mechanism for encoding expert knowledge. Conversely, an AV artifact's end-user environment can be seen as a place for decoding expert knowledge.

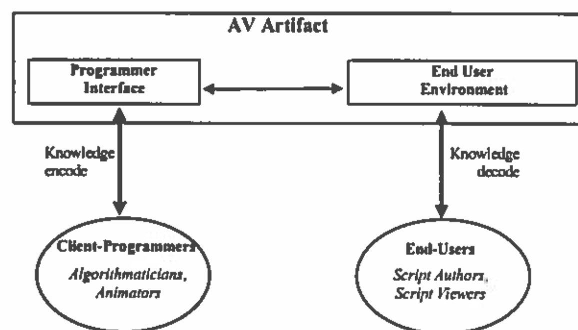



Figure 10. Knowledge Flow in Brown's (1988) User Model for AV Software

Given the clear avenues for knowledge flow provided by Brown's conceptual model, only two questions remain with respect to its adherence to the Knowledge Flow Assumption:

1. Which version of EF Theory (strong or one of the weak versions) does Brown's conceptual model support?

2. Must client-programmers necessarily be algorithm experts, and must end-users necessarily be learners? In other words, to what extent does Brown's (1988) conceptual model enforce a flow of knowledge *from* expert *to* learner?

To answer these questions, let us consider the activities that client-programmers and end-users typically take on.

*Client-programmers.* Client programmers either annotate algorithms to be visualized with markers "indicating interesting phenomena that should give rise to some type of display" (Brown, 1988, p. 12), or write code that maps those markers to AV views. Playing each of these roles requires a distinct form of expertise. Annotating an algorithm with event markers demands an understanding of just what is interesting about the algorithm. On the other hand, designing and implementing AV views calls for skill at computer programming, as well as detailed knowledge of the (algorithm animation system's) graphics library in terms of which AV views must be coded.

*End-users.* Prior to the execution of an AV, end-users may select (a) the particular AV views they wish to observe, (b) the input data on which the AV executes, and (c) the speed at which the AV is to execute. Using a *playback* interface that resembles that of a tape recorder, users may then alternatively start and pause the AV, altering the visible views and speed of execution at any point.

With respect to question (1), notice that end-users are afforded the opportunity to select input data and control AV execution. Therefore, they have the opportunity to be "involved" in the knowledge decoding process, and thus to build heightened attention. It follows that Brown's conceptual model supports the Learner Involvement version of EF Theory. With respect to question (2), notice that there exists an obvious disparity in the levels of expertise demanded of client-programmers and end-users; hence, as Brown (1988) points out, the two roles are played by different people in practice: "In an educational

setting, the course instructor and teaching assistants are usually the [client-programmers]" (p. 12), while the students are most often the end-users.[5] We see, then, that Brown's conceptual model indeed supports a flow of knowledge from expert to learner, just as the Knowledge Flow Assumption prescribes.[6]

Support for Algorithmic Fidelity

Recall that the first form of epistemic fidelity implied by EF Theory is fidelity with an algorithm's execution. As I shall illustrate below, three features of AV technology, which support what I call *direct generation*, *input generality*, and *typeset fidelity*, were clearly designed with algorithmic fidelity in mind.

---

[5]Brown's conceptual model provides for an additional kind of end-user—the *script author*. Much as user of a word processing system can record a macro, script authors may record their AV viewing sessions—including their choice of algorithms, views, input data, and execution speed—for future replay. Hence, an exception to this rule is that instructors often play the end-user role of *script author* in order to create custom videotapes for later exploration by their students. However, as a central feature of AV end-user environments, scripting has never really caught on; since Brown's BALSA system, not one system has supported it.

[6]In apparent contradiction to the Knowledge Flow Assumption, AV technology research has exhibited a clear interest, over the past ten years, in easing the job of the client-programmer. For example, Stasko (1989; Stasko & Kraemer, 1993) demonstrates the potential for a high-level programming framework tailored specifically to algorithm animation; several AV technologists (see, e.g., Duisberg,1987a; Helttula, Hyrskykari, & Raiha, 1989; Stasko, 1989; Mukherjea & Stasko, 1994; Stasko, 1990) explore techniques for designing and implementing AVs via direct-manipulation; Najork and Brown (1995) amalgamate a high-level animation library with an interpreted language; and Citrin and Gurka (1996) advocate the use of off-the-shelf morphing software to quickly produce algorithm animations from a sequence of static images. One might construe these attempts to ease the job of client-programmers as efforts to close the knowledge gap between client-programmer and end-user, and thereby to close the knowledge flow loop set up by Brown's conceptual model. However, AV technologists involved in this movement have made it clear that their motivation lies elsewhere. Indeed, rather than perceiving a need to abandon the Knowledge Flow Assumption, researchers of this persuasion have instead consistently cited the need to make AV technology easier to use for instructors (see esp. Citrin & Gurka, 1996, pp. 2–3; Stasko,1989, p. 10).

*Direct generation.* Noting humans' ineptitude when it comes to accurately describing dynamic algorithms, Baecker (1998) describes the obvious allure of AVs that are generated directly from executing algorithms:

> Unfortunately, a program's behavior cannot be described by a static drawing; it requires a dynamic sequence. . .It is difficult for us to enact these dynamic sequences directly. Our drawings are inaccurate. Our timing is bad. We make major mistakes, such as skipping and rearranging steps. Thus it would be useful to have animation sequences portraying the behavior of programs constructed automatically as a by-product of their execution, and therefore *guaranteed to portray this execution faithfully*. (p. 369, italics added)

The AV generation technique suggested by Baecker, which I call *direct generation*, has been nearly the only one considered by AV technologists (but see Citrin & Gurka, 1996). Indeed, so obvious seem its merits, and so tedious seem its manual alternatives, that direct generation has been a taken-for-granted constant in AV technology research ever since Baecker (1975) first advocated the technique for the production of instructional films on algorithms.

AV technologists have devoted significant time and effort to exploring alternative direct generation techniques. Early AV technology, including Brown's BALSA and Stasko's Tango, innovated so-called *annotative* techniques, in which client programmers annotate algorithm source code with interesting event procedures. When the algorithm is subsequently executed, those procedures both generate AV displays, and give rise to updates in them. The direct-manipulation techniques pioneered by Duisburg (1987a), Stasko (Mukherjea & Stasko, 1994; 1990), and Helttula (1989) are essentially variations on the annotative technique. Interesting event markers are inserted via direct manipulation, and the graphical updates to which they should give rise are specified using some combination of a graphics editor, dialog box fill-in, and direct manipulation. Finally, Roman *et al.* (1992) explore the potential for a declarative technique in which program-to-graphics mappings are specified by sets of rules. A fuller treatment of these direct generation techniques can be

found in Roman and Cox's (1993) taxonomy, which uses "means of direct generation" as a key dimension for classifying the extant AV technology.

Clearly, AV technology's universal support for direct generation follows from a perceived need for high epistemic fidelity with respect to an algorithm's execution. As Baecker points out above, AVs produced by humans tend to be inaccurate; that is, they tend to have low algorithmic fidelity. Thus, if algorithmic fidelity were not such a concern for AV technologists, manually-generated AVs would suffice; direct-generation AV technology would not be necessary.

*Input Generality.* The generation of AVs directly from implemented computer algorithms implies that AVs, just like the algorithms on which they are based, *must operate on general input.* As Stasko (1989) points out, while it makes the client-programmer's job more difficult, such *input generality* is necessary to preserve algorithmic fidelity:

> [A]lgorithm animations. . .are especially difficult because they must proceed given any set of input data from the program. They are not one-shot animations that can be fine-tuned and discarded. They must be general enough that their animation actions convey the program's meaning for all of its possible executions. (p. 7)

Input generality is directly supported by Brown's (1988) notion of the *input generator*, which allows a user to specify interactively input data for an AV. Although they have not always accepted input data through a graphical-user interface like BALSA's, subsequent AV systems have consistently supported the creation of AVs that run on general input. For example, Stasko's Tango (Stasko, 1989) and Polka (Stasko & Kraemer, 1993) systems provide support for general-input AVs in the form of parameterized animation routines, which obtain their inputs directly from the underlying algorithm driving the animation.

As Stasko points out in the above quote, general-input AVs are particularly challenging to write. If AV technologists were not concerned with algorithmic fidelity, they would not be so intent on supporting general-input AVs. Instead, it seems reasonable that

they would shift their focus to providing explicit mechanisms for *restricting* the range of input data. While such mechanisms would limit an AV's algorithmic fidelity, they would hold promise in simplifying the job of the AV writer—an important item on AV technologists' research agenda, as we saw above.

*Typeset fidelity.* Nearly all extant AV technology requires that AVs be programmed in terms *of quantitative graphics.* By quantitative graphics, I mean a graphical programming system in which (a) object position must be specified in terms of Cartesian coordinates, and (b) object attributes, as well as animation, must be specified in terms of real or integer data. The mapping of executing algorithms to quantitative graphics leads to the kinds of high-quality drawings that one would expect to find in an algorithms textbook. Because they resemble the illustrations found in typeset manuscripts, such drawings have what I call *typeset fidelity*. According to EF Theory, AVs with high typeset fidelity are attractive not only because they have the polished look of textbook illustrations (and hence *cultural fidelity*, as explained in the following subsection), but also because they are "always accurate, even the ones which would tax the best of draftsmen" (Brown, 1988, p. 5).

As was the case for direct generation and input generality, typeset fidelity does not come without formidable implementation costs for the AV designer. Unlike computers, humans do not necessarily think about graphics in terms of Cartesian coordinates; hence, laying out AVs in terms of Cartesian coordinates is a potentially time-consuming endeavor, especially when compared to, say, creating pen-and-paper sketches of AVs. That AV technologists have nonetheless insisted on facilitating the creation of precise, polished, high-typeset fidelity illustrations, rather than imprecise, scruffy, low-typeset fidelity illustrations clearly indicates an allegiance to EF Theory.

Support for Cultural Fidelity

The second form of fidelity implied by EF Theory is fidelity with the accepted conventions and nomenclature of expert algorithmaticians; one might call this fidelity with "the culture of AV." Undoubtedly inspired by illustrations in algorithms textbooks, AV technologists have been in a continuous process of developing this culture through their technology-building efforts. For example, Baecker's (1981) *Sorting Out Sorting* film firmly established the *sticks* and *dots* views of sorting algorithms; nearly two decades later, algorithmaticians refer to such views as "canonical." Building on Baecker's work, Brown and Sedgewick (1985) developed an expanded suite of AVs for sorting, searching, graph, and computational geometry algorithms; these AVs have subsequently appeared in various versions of Sedgewick's algorithms textbook (Sedgewick, 1988).

In reviewing the AV technology literature, one cannot help but notice that the views and techniques originally established by Baecker, Brown, Sedgewick, and other pioneers of AV technology have been consistently supported and extended by AV technology ever since (see, e.g., Brown & Hershberger, 1992; Duisberg, 1987b; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1989; Stasko & Kraemer, 1993). In perhaps the only attempt to document this emerging culture of AV, Brown and Hershberger (1992) concisely review the characteristics of views (state cues, history, animation transition types), and presentation techniques (multiple views, judicious input data selection, side-by-side algorithm comparison) that extant AV technology has consistently supported. That AV technologists have demonstrated a commitment to supporting the views and presentation techniques of the past suggests that the ability to support the established AV culture has become a *de facto* requirement among AV technologists.

Two lines of AV technology research, in fact, embrace this requirement explicitly. Lens (Mukherjea & Stasko, 1994) is a visualization system that allows one to create AVs rapidly through a combination of dialog box fill-in and direct manipulation. In designing Lens's visualization language, Mukherjea and Stasko (1994) studied 42 expert AVs drawn from several problem domains, including sorting, searching, and graph theory. In a similar vein, Douglas, Hundhausen, and McKeown (1995) and Chaabouni (1996) staged empirical studies involving a total of 24 algorithm experts; their interest was in designing AV languages from a user-centered (i.e., expert) perspective.

## Influence on Effectiveness Evaluation

Nowhere is EF Theory's influence more obvious than in the arena of effectiveness evaluation. Indeed, it is through such evaluation that researchers are forced to make explicit their theory of effectiveness, which is bound up in the particulars of an empirical evaluation's design. The designs of ten controlled experiments that have explicitly tested AV's influence on learning (Byrne, Catrambone, & Stasko, 1996, §2 & 3; Kann, Lindeman, & Heller, 1997; Lawrence, 1993, ch. 4–9; Stasko, Badre, & Lewis, 1993) bear a remarkable resemblance. Below, I illustrate the way in which EF Theory has guided the design of these experiments.[7]

The Knowledge Representation Assumption asserts that individuals carry around symbolic representations of algorithms in their head. Thus, EF Theory holds that the appropriate unit of analysis for any effectiveness evaluation is individual knowledge.

---

[7]Gurka (1996; Gurka & Citrin, 1996) proposes a series of refinements that she claims will make experiments more sensitive to the pedagogical benefits of AV technology. However, rather than fundamentally altering the design of the experiments discussed in this section, her refinements appear simply to tweak surface features of their design.

Accordingly, in all AV effectiveness experiments, participants have worked with AV technology individually, and their individual knowledge has been the focus of evaluation.

In particular, in line with the Knowledge Flow Assumption, effectiveness has always been operationalized in terms of the acquisition of target knowledge structures, which AV viewers are assumed to glean from learning sessions in which they are exposed to both AVs and alternative media (e.g., books, articles). Researchers have taken individual performance on a written post-test as evidence for the successful transfer of such structures, which are often classified as either *declarative* (i.e., what the program does) or *procedural* (i.e., how the algorithm works).[8]

By comparing the efficacy of various pedagogical exercises involving AV technology, all ten experiments demonstrate an implicit commitment to the Graphical Medium Effectiveness Assumption. Four of the experiments, in fact, assert the significance of the graphical medium as a causal factor in knowledge acquisition (Byrne, Catrambone, & Stasko, 1996, sec. 2 & 3; Kann, Lindeman, & Heller, 1997; Stasko, Badre, & Lewis, 1993). To see this, one need only consider their between-subjects design, as illustrated Figure 11. The underlying assumption of these experiments' design is that alternative learning *media* (see the "Expose" column in Figure 11) are more or less effective in the transmission of the target knowledge, which is subsequently quantified (see the "Measure" column Figure 11) and compared (see the "Compare" column in Figure 11).

---

[8]In fact, researchers have been keenly interested in ascertaining just what type of knowledge AVs successfully transfer; see, e.g., (Lawrence, 1993).

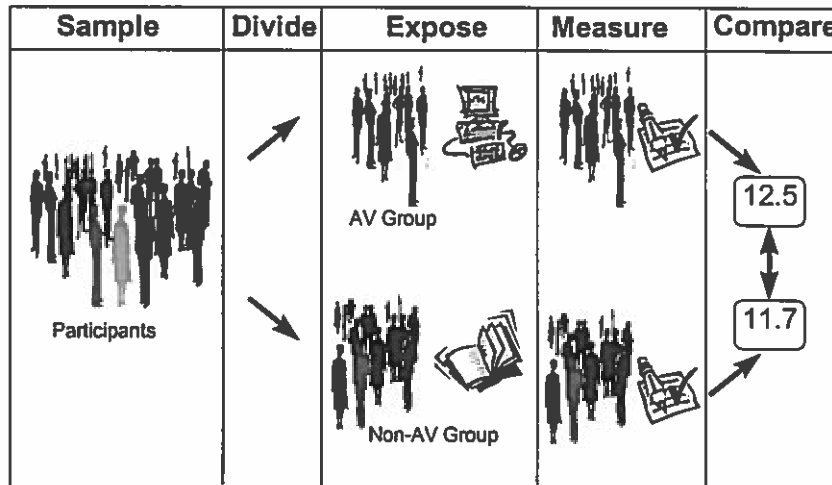



Figure 11. Prototypical Design of an AV Effectiveness Experiment

Finally, all experiments operationalize effectiveness in terms of *the robustness of individual knowledge acquisition*—the extent to which individuals acquire the target knowledge encoded by the AVs and alternative artifacts. In addition, at least one of Lawrence's (1993, ch. 5) experiments considers *knowledge acquisition time* as a dependent variable. Researchers' past interest in the robustness and speed of knowledge transfer clearly reflects a commitment to the EF Theory, which views robust knowledge transfer and efficient knowledge acquisition as the principal benefits of high-epistemic fidelity AVs.

## Influence on Pedagogy

As I have already argued, several features of AV technology uphold EF Theory. One of those features, the dyadic user model (see Figure 10, p. 36), accommodates a set of pedagogical practices firmly rooted in EF Theory. The dyadic user encourages teachers and students to play distinct roles. Serving as client-programmers, teachers design and implement AVs. Students, on the other hand, are the end-users, viewing and interacting

with those AVs. According to the Knowledge Flow Assumption, teachers *encode* the target knowledge to be learned, while students *decode* and internalize that knowledge. Its traditional advocacy of such an "encode/decode" model of learning constitutes the strongest evidence of past AV pedagogy research's adherence to EF Theory. Specifically, we find the encode/decode model of learning, as well as the EF-based assumptions underlying it, well intact in lectures, labs, and individual study—three AV-based pedagogical treatments recommended by algorithm educators.

AV in Lectures

Brown University's Electronic Classroom project pioneered the use of AV technology as a lecture aid. Brown University's Foxboro Auditorium, the original electronic classroom, contained a high-speed network of 60 high-performance graphical workstations. Using the network, instructors could broadcast AVs running on their machine to all student machines. As Brown (1988) explains, this medium of communication provided an attractive alternative to more traditional teaching media: "Rather than using chalkboards or viewgraphs to show static diagrams—often incorrectly drawn and messy at best—or asking teaching assistants to become Thespians in order to 'enact' procedure calls, searching and sorting algorithms, traveling salesmen, and so forth, dynamic simulations of the algorithms and programming concepts are presented via the workstations" (p. 165).

The structure of Brown's electronic classroom places the instructor "on stage"; each lecture can be seen as a performance in which students watch algorithm simulations both created and narrated by the instructor. As Brown University instructors note in a later experience report (Bazik, Tamassia, Reiss, & van Dam, 1998), "without interaction [among the instructor, students, and AV software], . . . a demonstration differs little, pedagogically,

from a film or videotape" (pp. 3–4). It follows that, as an adjunct to such "sage-on-stage" lectures, AV technology serves as a medium for transmitting knowledge—very much in accordance with the Knowledge Flow Assumption.[9]

AV in Labs

The Denning Committee report (Denning, 1989), which recommends a core computer science curriculum, emphasizes the importance of a laboratory experience that should complement course lectures by allowing students to explore difficult concepts through experimentation. Recognizing the potential for AV technology as a tool in such a lab experience, Naps (1990) develops a pedagogy for so-called "algorithm visualization laboratories." The first principle of the pedagogy, which has been adopted by several computer science instructors since its inception, is that students should not have to implement the algorithms explored during labs. Students may have to modify an instructor-provided algorithm, but "such programming is never an end itself"; rather, "it is part of an experiment being done in the laboratory" (p. 105). A second principle of the pedagogy is that *concepts*, rather than code, should be emphasized in the lab. Implicit in this principle is the idea that AV software is ideally suited for that purpose. As Naps puts it,

> [t]he student should be able to visualize and explore the conceptual foundations of algorithms instead of the code behind them. . .[T]he amount of pure text (either code or output) seen by the student during a lab should be minimized. . .[A] lab workstation [running AV software] should be used to enhance the student's mental visualization of how the algorithm manipulates critical data structures. (p. 106)

---

[9]According to several experience reports, the value of AV technology as a lecture aid rests in its ability to assist instructors in providing convincing on-the-spot answers to students' questions (see, e.g., Bazik, Tamassia, Reiss, & van Dam, 1998; Brown, 1988, Appendix A; Gurka & Citrin, 1996). Notice that here, too, the Knowledge Flow Assmption is at work; the only difference is that the flow of knowledge can be more responsive to students' "knowledge gaps" than it can be in non-interactive lectures.

In the AV laboratories described by Naps, the mode of interaction clearly departs from that of classroom teaching; instead of being fed with knowledge, students "explore" concepts by engaging in "experiments." However, the rationale underlying this departure from classroom teaching, as articulated by Naps in the above quote, coincides with the EF-motivated rationale behind using AV technology in lectures: Students' exposure to AVs will "enhance [their] mental visualization." The reason behind such enhancement, one can infer, is that pure textual representations of algorithms are not conceptual, while visual representations (i.e., AVs) are. Since AVs are seen to provide a closer match with target (conceptual) knowledge structures, they are to be preferred to text as the medium for target concept transfer in laboratories.

AV for Individual Study

A third pedagogical use of AV technology is for individual study, in which students explore AVs on their own. For example, the CD-ROM (Gloor, 1992) that accompanies the popular algorithms textbook of Cormen, Leiserson, and Rivest (1990) is ideally-suited for individual study. As Gurka and Citrin (1996) point out, providing students with AVs for individual study has both advantages and disadvantages:

> AV study aids are an improvement over unsupported studying, because *correctness* of the visualization is ensured, while allowing the student to produce "what-if" scenarios. . .If students are allowed complete exploratory learning, in which they make most of the choices about examples used and their sequence, they may develop an *incorrect* or *incomplete model due* to selecting a biased problem set.
> (p. 183, italics added)

Once again, the rationale for AV technology's use as a study aid appears to be mired in EF Theory. One the one hand, AV is seen to provide a higher-fidelity depiction of an algorithm than other study aids. On the other hand, giving students too much freedom

might lead them astray, since they may choose a "biased" problem set, leading to low epistemic fidelity AVs that cause students to acquire the wrong knowledge.

## A Critique of EF Theory

The foregoing review of the AV technology literature highlights several ways in which EF Theory has quietly guided AV technologists, evaluators, and educators. To the extent that EF Theory has influenced the design, evaluation, and use of AV technology, any deficiencies in the theory have the potential to impede further research. Below, I use a review of past empirical research into AV effectiveness as a basis for critiquing the validity of the theory. As I shall illustrate, the predictions of EF Theory are not borne out by the results of past studies.

At least ten controlled experiments have explicitly evaluated the efficacy of various pedagogical treatments involving AV technology (Byrne, Catrambone, & Stasko, 1996, §2 & 3; Kann, Lindeman, & Heller, 1997; Lawrence, 1993, ch. 4–9; Stasko, Badre, & Lewis, 1993). Table 3 (next page) provides a synopsis. For each experiment cited in column 1, the pedagogical treatments that were compared appear in column 2; the measures (dependent variables) appear in column 3; and a summary of the experiment's key results appears in column 4.

Although the experiments' treatment groups appear quite varied on the surface, further analysis indicates that all of the experiments actually manipulate just five independent variables. Table 2 (p. 52) presents those independent variables, along with the values of the variables considered by the experiments.

Table 1. Summary of Ten AV Effectiveness Experiments

| Study | Pedagogical Treatments | Dependent Measures | Key Results |
|---|---|---|---|
| (Stasko, Badre, & Lewis, 1993) | 1. Study text-only<br>2. Study text + view animation | 1. Post-test accuracy | No significant difference |
| (Lawrence, 1993, ch. 4.4) | Nine treatment groups, all of which studied text and then viewed one of nine different animations formed by varying two factors:<br>1. Data set size (16, 27, or 41 elements)<br>2. Data rep. (vert. bars, horiz. bars, dots) | 1. Post-test accuracy | No significant differences |
| (Lawrence, 1993, ch. 5) | Four treatment groups, all of which studied text and then viewed two animations formed by varying two factors:<br>1. Order of presentation<br>2. Representation style (labeled or unlabeled)<br>(Note that spatial and verbal abilities were treated as covariates) | 1. Post-test accuracy<br>2. Time to take post-test | 1. No significant differences<br>2. Spatial and verbal abilities were not correlated with performance |
| (Lawrence, 1993, ch. 6) | 1. Study text + passively view animation<br>2. Study text + actively view animation (by constructing own input data sets) | 1. Post-test accuracy<br>2. Time to take post-test | Participants who actively viewed animation scored significantly higher than students who passively viewed animation |
| (Lawrence, 1993, ch. 7) | Four treatment groups, all of which studied text and then viewed animations formed by varying the following two factors:<br>1. Representation style (color vs. black-and-white<br>2. Representation style (algorithmic step labels vs. no labels) | 1. Post-test accuracy<br>2. Accuracy on a transfer task | 1. Participants who viewed black-and-white animations scored significantly higher on post-test<br>2. Participants who viewed labeled animations scored significantly higher on post-test |
| (Lawrence, 1993, ch. 8.2) | Four treatment groups formed by varying the following two factors:<br>1. Order of presentation (text first or animation first)<br>2. Order of presentation (selection sort first vs. Kruskal MST first) | 1. Post-test accuracy | No significant differences |
| (Lawrence, 1993, ch. 9) | 1. Lecture-only<br>2. Lecture + passively view animation<br>3. Lecture + actively view animation (by constructing own input data sets) | 1. Free-response post-test accuracy<br>2. Multiple choice/true-false post-test accuracy | On free-response post test, participants who heard lecture and actively viewed animation significantly outperformed students who only heard lecture |
| (Byrne, Catrambone, & Stasko, 1996, §2) | 1. Study text only<br>2. Study text + make predictions<br>3. Study text + view animation<br>4. Study text + view animation + make predictions | 1. Post-test accuracy<br>2. Prediction accuracy | Participants who viewed animation and/or made predictions scored significantly higher on hard" questions than participants who did neither |
| (Byrne, Catrambone, & Stasko, 1996, §3) | Same as previous | Same as previous | Same as above, but difference was detected for "procedural" questions |
| (Kann, Lindeman, & Heller, 1997) | 1. Program algorithm<br>2. Program algorithm + construct animation<br>3. Program algorithm + view animation<br>4. Program algorithm + view animation + construct animaton | 1. Programming accuracy<br>2. Post-test accuracy | Participants who viewed animation scored significantly higher on post-test than participants who did |

Table 2. Independent Variables Manipulated in Ten AV Effectiveness Experiments

| Independent Variable | Values Considered |
|---|---|
| Learning medium | 1. Text-only<br>2. Text-and-animation |
| Learning medium order | 1. Text first<br>2. Animation first |
| Cognitive and Spatial Ability | 1. Spatial ability (according to a paper-folding test)<br>2. Verbal ability (according to a vocabulary test) |
| Animation representation | 1. Data set size (9, 25, or 41 elements)<br>2. Data representation (horizontal sticks, vertical sticks, dots)<br>3. Hue (color, black-and-white) |
| Representation redundancy | 1. Data element labeling (labels, no labels)<br>2. Algorithm step labeling (labels, no labels) |
| Level of learner involvement | 1. Passively view animation<br>2. Program algorithm while viewing animation<br>3. Actively view animation (construct own input data sets, make predictions)<br>4. Construct animation |

Notice that each of these independent variables coincides with a particular version of EF Theory. In particular, the experiments that manipulated learning medium, learning medium order, and animation representation aimed to support Strong EF Theory; the experiments that compared labeled and unlabeled representations aimed to support the Dual-coding version of Weak EF Theory; the experiment that considered spatial and verbal abilities aimed to support the Individual Differences version of Weak EF Theory; and the experiments that manipulated level of of learner involvement aimed to support the Active Learner Involvement version of Weak EF Theory. Table 3 reorganizes the experiments according to the versions of EF Theory they were designed to support. For each version of EF (column 1), the table lists the corresponding independent variables (column 2) and supporting experiments (column 3).

Table 3. Ten AV Effectiveness Experiments vis-à-vis the Version of EF Theory For Which They Were Designed to Provide Evidence

| EF Theory version | Independent Variable(s) | Supporting Experiments |
|---|---|---|
| Strong EF | Learning medium<br>Learning medium order<br>Animation representation | 1. (Stasko, Badre, & Lewis, 1993)<br>2. (Lawrence, 1993, ch. 4)<br>3. (Lawrence, 1993, ch. 5)<br>4. (Lawrence, 1993, ch. 7)<br>5. (Lawrence, 1993, ch. 8) |
| Individual differences | Cognitive and spatial ability | 1. (Lawrence, 1993, ch. 5) |
| Dual-Coding | Representation redundancy | 1. (Lawrence, 1993, ch. 5)<br>2. (Lawrence, 1993, ch. 7) |
| Learner involvement | Level of learner involvement | 1. (Lawrence, 1993, ch. 6)<br>2. (Lawrence, 1993, ch. 9)<br>3. (Byrne, Catrambone, & Stasko, 1996 295, §2)<br>4. (Byrne, Catrambone, & Stasko, 1996 295, §3)<br>5. (Kann, Lindeman, & Heller, 1997) |

Figure 12 takes the analysis one step further by plotting the proportion of "successful" experiments vis-à-vis each EF Theory version. In other words, Figure 12 illustrates the proportion of experiments that detected statistically significant results in support of each EF Theory version. Insofar as those significant differences substantiate the causality of the independent variables manipulated in each experiment, and insofar as those independent variables express the underlying assumptions of each respective EF Theory version, the proportions plotted in Figure 12 furnish a crude measure of each EF Theory version's experimental support. [10]

---

[10]Note that meta-analytic techniques like those proposed in (Hedges & Olkin, 1985) could be used to determine the *effect size* of each independent variable—a more reliable indicator of experimental support. However, such an analysis is beyond the scope of this thesis.

As Figure 12 suggests, the evidence in support of each of the various versions of EF Theory varies considerably. Only one of the five experiments designed to support Strong EF Theory yielded a significant result; in all others, merely manipulating some aspect of the representation had no bearing on learning outcomes. Similarly, the Individual Differences version of Weak EF Theory was tested in one of Lawrence's (1993, ch. 5) early experiments, and then abandoned.

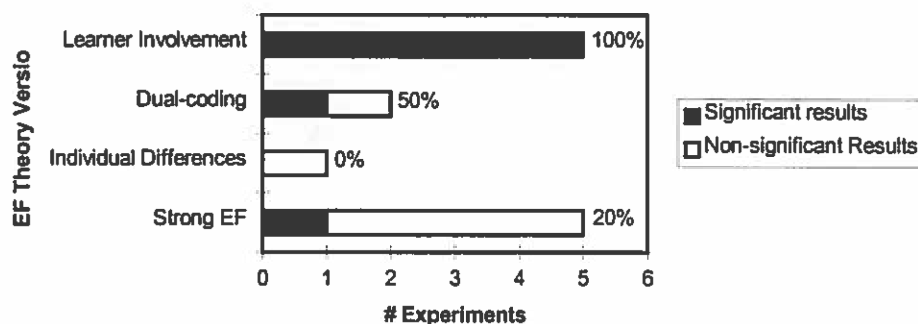



Figure 12. Summary of the Experimental Support for each Version of EF Theory

The Dual-Coding version of Weak EF Theory appears somewhat more promising than both Strong EF and the Individual Differences version of Weak EF Theory. In particular, Lawrence (1993) found that animations in which data elements are redundantly labeled do not lead to better post-test performance, whereas animations in which the algorithm's *conceptual steps* are redundantly labeled do lead to higher post-test performance. This latter result replicates an earlier experiment involving animated explanations of physical systems (Mayer & Anderson, 1991). In that experiment, animations that contained simultaneous narratives led to higher student performance on a post-test than did animations without such narratives.

Finally, the Learner Involvement version of EF Theory appears to be the most promising of the theories; indeed, it is the only one that is consistently supported by

experimental evidence. In all five experiments that put this version of the theory to the test, students who were actively involved—either by programming the algorithm while viewing an animation, creating their input data and viewing an animation, or making explicit predictions while viewing an animation—performed significantly better on post-tests than those who passively watched animations (Byrne, Catrambone, & Stasko, 1996, §2 & §3; Lawrence, 1993, ch. 6), or those who were not given the opportunity to watch animations at all (Kann, Lindeman, & Heller, 1997; Lawrence, 1993, ch. 9).

Lending further credence to the Learner Involvement version of EF Theory is the historical evolution of empirical studies and experiments involving AV technology (see Figure 13). The early studies and experiments of Badre et al. (1991), Stasko, Badre, and Lewis (1993), and Lawrence (1993, ch. 4 and 5) explored Strong EF Theory. However, a string of non-significant results led to the theory's ultimate abandonment early on in Lawrence's (1993) dissertation research. While Lawrence (1993) intermittently considered the Individual Differences (ch. 5) and Dual-Coding (ch. 5 and 7) versions of Weak EF Theory, her experiment on viewer activity (ch. 6) marked a definite turning point in the evolution of AV empirical studies, for it yielded a significant result by explicitly manipulating Learner Involvement. The viability of the Viewer Involvement version of Weak EF Theory was thus established.

From that point on, every single experiment (Byrne, Catrambone, & Stasko, 1996; Kann, Lindeman, & Heller, 1997) and empirical study (see, e.g., Kehoe & Stasko, 1996; Wilson, Katz, Ingargiola, Aiken, & Hoskin, 1995) of AV technology has included learning sessions with active learner involvement. In fact, as we shall see in the next chapter, some of the most recent AV pedagogy research (Stasko, 1997) has been interested in pushing active learner involvement even further.

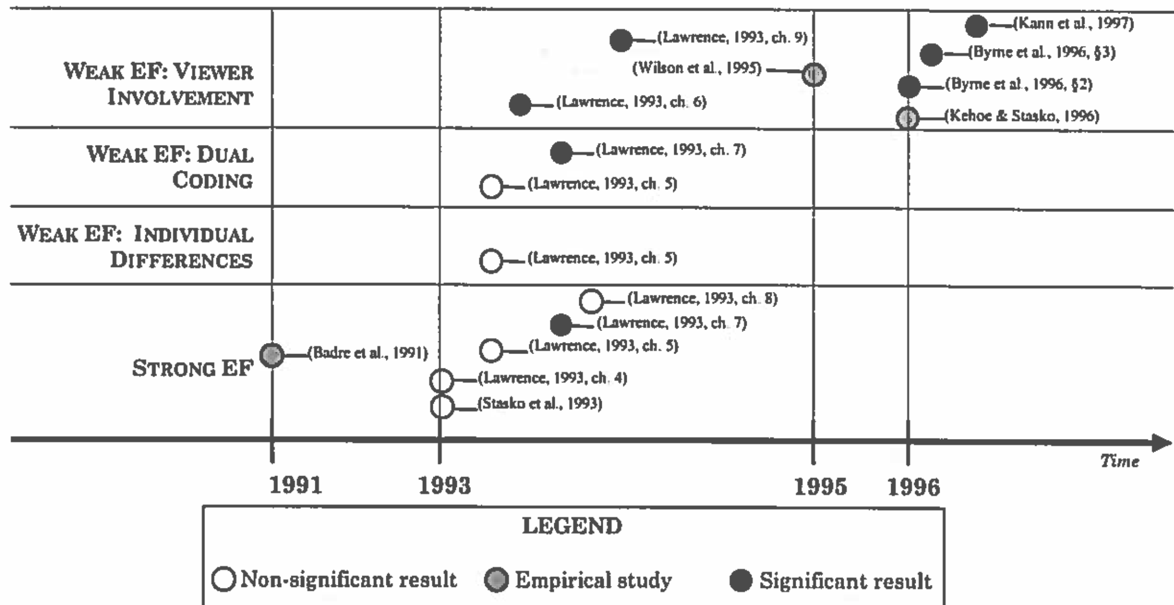



Figure 13. The Historical Evolution of AV Effectiveness Experiments and Empirical Studies

In sum, the Learner Involvement version of Weak EF Theory has garnered by far the most consistent empirical support; the historical trend in AV effectiveness studies toward active viewer involvement provides additional support for that version of the theory. Given that what AV viewers *do* has been the most significant factor in past effectiveness experiments, one must question EF Theory's assumption that an AV's epistemic fidelity matters. Indeed, the experimental results raise the possibility that an AV's epistemic fidelity matters far less than what the learner does with the AV.

## Summary

In this chapter, I have described Epistemic Fidelity Theory, a particular account of what knowledge is, how it is acquired, and why AV technology is effective in facilitating knowledge acquisition. By articulating the implications of EF Theory and observing their

influence on the published literature, I have shown EF Theory's stronghold on past research into AV technology, evaluation, and pedagogy. Finally, in scrutinizing the experimental support for EF Theory, I have found definite reason to be concerned. Contrary to EF Theory's predictions, the Learner Involvement version of Weak EF Theory has garnered more consistent experimental support than any other version of EF Theory, calling into question the putative value of epistemic fidelity, and suggesting that a fundamental rethinking of our research agenda is in order. That rethinking, which I begin in the next chapter, must necessarily begin with a shift in theoretical orientation. Indeed, if we continue to allow the assumptions of EF Theory to guide our explorations of AV technology, we are likely to continue to be disappointed with the discrepancy between our intuition, which says that AV technology should be pedagogically effective, and experimental results, which indicate that what learners do matters more than what learners see.

# CHAPTER III

## CONSTRUCTIVISM: AN ALTERNATIVE THEORETICAL FOUNDATION

> The way we segment the flow of our experience, and the way we relate the pieces we have isolated, is and necessarily remains an essentially subjective matter. Hence, when we intend to stimulate and enhance a student's learning, we cannot afford to forget that knowledge does not exist outside a person's mind.
>
> (von Glasersfeld, 1996, p. 5)

> Engaging in practice, rather than being its object, may well be a condition for the effectiveness of learning. (Lave & Wenger, 1991, p. 93)

In the previous chapter, I illustrated that the extent to which learners are actively involved in the visualization process has the most significant impact on successful learning outcomes. A logical starting point in the search for an alternative theoretical foundation, then, is to consider pedagogical alternatives that get students *even more involved* in the visualization process.

One obvious alternative is to have students construct their own AVs, rather than having them interact with AVs constructed by someone else. While not motivated by learning theory, a few computer science educators have already explored so-called "animation assignments," in which students are asked to construct their own AV of a given algorithm. For example, Brown (1988, Appendix A) describes animation assignments used in past offerings of Brown University's undergraduate algorithms course. For such assignments, students were provided with a collection of animation routines that they could insert into their programs in order to animate them. The animation routines allowed students to generate animations of their programs that resembled the ones presented in the lectures.

In contrast, Stasko (1997) has developed a high-level interpreted language called *Samba*, which is designed specifically to support students in their algorithm animation-building efforts. In past offerings of his third-year undergraduate algorithms course, Stasko used Samba as the basis for animation assignments. In these assignments, students developed animations of fundamental sorting and graph algorithms "that would help explain the algorithm[s] to a person not familiar with [them]" (p. 6). In his assessment of animation assignments, Stasko is quite positive. He notes that students appear not only to enjoy building their own animations, but also to gain increased understanding as a result.

Recall that the main argument of this thesis is that, in order to harness the pedagogical promise of algorithm visualization, we need to rethink our underlying theory of effectiveness. What theoretical foundation would predict the pedagogical shift both suggested by the meta-analysis of the last chapter, and recently explored by a few computer science educators?

In this chapter, I propose *constructivism* as the theoretical position that best accounts for the pedagogical shift toward getting students more involved in the visualization process. As with EF Theory, there are multiple versions of this theory. I begin by introducing the cognitive version, which, because it focuses on individual knowledge, bears some resemblance to EF Theory. However, I argue that the target skills taught in an algorithms course are inherently *social*. Consequently, an alternative version of the theory, *sociocultural constructivism*, is actually more appropriate because it situates learning in the social environment in which it occurs. In the second part of the chapter, I discuss sociocultural constructivism and identify the implications it has for the design, evaluation, and pedagogical use of algorithm visualization technology. As we shall see, if we are to take sociocultural constructivism seriously, we need to go beyond having students construct their own AVs: We need to encourage students to participate within a community that *practices*

*algorithms*, which fundamentally entails the construction, presentation, and discussion of AVs. To explore and validate the potential of this theoretical shift requires a naturalistic field study—an endeavor I take up in Chapter IV.

## Cognitive Constructivism

Inspired by Piaget's research into childhood development (see, e.g., Gruber & Voneche, 1977), cognitive constructivism encompasses both an epistemological framework and learning theory that differ markedly from those of the EF view. Like the EF view, cognitive constructivism views knowledge at the level of the individual learner. However, rather than regarding knowledge as representations of an objective reality that people carry around in their heads, cognitive constructivism asserts that there is no absolute knowledge. Instead, cognitive constructivism holds that individuals *construct* their own individual knowledge "out of the bewildering array of sensations which have no order or structure besides the explanations. . .which [humans] fabricate for them" (Hein, 1991).

This epistemological position gives rise to a theory of learning that differs fundamentally from that of EF Theory. In particular, it implies that knowledge cannot merely be transferred, in the EF sense, from person to person by means of visual representations. Instead, knowledge must be individually (re-)constructed by the individual. Learning as a knowledge construction process, as well as the way in which this process changes as a learner grows, form key assumptions of the cognitive constructivist view. Below, I discuss these assumptions, as well as their implications for the design, evaluation, and pedagogical use of AV technology.

## Key Assumptions

### Learners Actively Construct Their Own Knowledge

The first key assumption of the cognitive constructivist view is an epistemological one, and is perhaps best understood by way of contrast with EF Theory. On the EF View, there exists a deterministic relationship between an objective reality and individual knowledge about that reality. On the cognitive constructivist view, in contrast, knowledge is an individual creation; it is constructed by each learner for herself. There is thus no notion of absolute knowledge that is somehow represented in people's heads.

How do learners construct their own knowledge? According to cognitive constructivism, in the process of knowledge construction, learners assimilate their experiences in the world into their existing understandings. In other words, learners actively construct new understandings by interpreting new experiences within the context of what they already know (see, e.g., Resnick, 1989).

### Stages of Intellectual Development: Concrete to Abstract

A second key assumption of cognitive constructivism follows from Piaget's theory of childhood development (see, e.g., Gruber & Voneche, 1977). Piaget's theory postulates that children's intellectual development progresses through different stages, beginning with concrete thinking, and ultimately advancing to formal thinking. These stages are held to be genetic or innate; they are not formally taught. Moreover, they have implications for the ways in which learners construct meanings. When learners are in the concrete stage of thinking, which is well under way at age 6, their conceptual structures—that is to say, the

knowledge into which they assimilate their experiences—are firmly grounded in the physical world. Later on, in the formal stages of thinking (which begins around age 12), learners' conceptual structures are more abstract, enabling them to construct knowledge through more abstract ways of thinking. For example, they may use the concept of variables to reason about the possible states of a physical system, or they may use meta-knowledge (knowledge about knowledge) to reflect on their own thought processes (see, e.g., Papert, 1980, ch. 1).

## Implications for Pedagogy

What implications do these assumptions have for the ways in which computer science educators ought to use AV as a pedagogical aid? The first assumption implies that instructors should not use AV technology to facilitate lectures or demonstrations in which students are members of a passive audience. Instead, the theory suggests that computer science educators ought to use AV technology as a means of getting students actively involved in their own meaning-construction processes; AV technology should be a basis for *active learning.*

Active learning, on the cognitive constructivist view, fundamentally entails self-directed engagement with one's environment. Here, environment is not meant in the literal sense, as one's physical surroundings. Rather, in the cognitive constructivist sense, environment is quite subjective, encompassing the "cognitive and perceptual structures" that one has abstracted and constructed from one's experiential world (von Glasersfeld, 1996, pp. 4–5). Active learners choose their own problems based on their interests and motivation; construct conceptual and physical objects to represent the problems; manipulate the objects; observe the effects; and, through this process, construct new understandings. Along the way, learners are bound to formulate false theories—that is, incorrect understandings of what is

to be learned. However, according to cognitive constructivism, the development of such false theories is an important step in the knowledge-construction process. In fact, the theory suggests that it is foolish to force-feed learners the correct theories; students must ultimately *invent* the correct theories for themselves.

If computer science educators are to take active learning seriously, they should encourage students to use AV technology as a tool for discovering for themselves how algorithms work. In the previous chapter, I showed a trend, in AV-based pedagogical approaches, toward "active viewer involvement": having learners explore a visualization by selecting their own input data sets and observing the execution of the animation for those data sets. While this learning paradigm is certainly a step in the right direction, cognitive constructivism would hold that it does not go far enough, because it fails to involve learners in the process of defining and constructing the objects of the visualization. Indeed, the cognitive constructivist view would maintain that, in order to reap the most learning benefits from an AV, students must construct the AV for themselves. Note that this is precisely the case in the AV construction assignments described above.

The second assumption, which follows from Piaget's child development theory, establishes a theoretical basis for the pedagogical use of AVs. Papert (1980), among others, has observed that the concrete and formal stages of learning of which Piaget speaks can be bridged through computer-implemented models that stand in analogy to the abstract concepts to be learned. As Papert puts it, "computer models [seem]. . .able to give concrete form to areas of knowledge that. ..[appear]. . .so intangible and abstract" (Papert, 1980, p. 23). On the cognitive constructivist view, then, AVs are pedagogically valuable precisely because they model an abstract process in terms of concrete objects, thus establishing a basis on which students can progress from the construction of concrete knowledge (AVs as physical objects) to the construction of abstract knowledge (AVs as general models of algorithms).

## Implications for Technology Design

As discussed in Chapter II, the dominant user model for AV software encourages a division of labor between instructors and students. Instructors are generally the ones who design and implement AVs, and learners are generally the ones who view and interact with AVs. Given its position that learners actively construct their own knowledge, cognitive constructivism suggests a break from this model: AV technology should be designed for learners, not instructors. In other words, AV technology should be designed such that learners are the ones who design, implement, view, and interact with AVs.

What, specifically, does this implication have to say about design? The fact that learners are the ones who design and implement AVs suggests that the models that AV systems define for AV specification must be carefully adapted to the needs and abilities of learners. In many extant AV systems (e.g., Brown, 1988; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1989), AV creation requires the use of relatively low-level graphics packages. While one would expect an expert instructor to be able to handle low-level graphics, it would be unreasonable to expect learners to be able to program an AV in terms of low-level graphics.

To empower learners to construct their own AVs, AV systems should support AV creation environments that consists of objects—both physical and conceptual—with which learners are likely to be familiar. Indeed, according to the cognitive constructivist view, unless learners are able to represent algorithms in terms of familiar objects, they will be unable to bridge the gap between the concrete and the abstract. As a result, AV technology will lose its value as a tool with which learners can think, experiment, and construct new understandings.

## Implications for Effectiveness Evaluation

According to cognitive constructivism, in order to set up the best possible conditions for learning, effective learning technology should make available the right conceptual and physical tools—those that will make it possible for learners to build their own understandings. In the case of algorithms, what would the desired understandings be? That is, what knowledge and skills would one expect learners to take away from their experiences with AV technology?

Recall that EF Theory sees knowledge as symbolic structures in the heads of individuals. It is common, in fact, for EF-minded psychologists to distinguish between two kinds of symbolic knowledge: "declarative" knowledge (facts), and "procedural" knowledge (skills). As we saw in Chapter II, the post-tests used in past studies of AV effectiveness were designed specifically to measure learners' acquisition of these two forms of knowledge. In the case of an algorithm, declarative knowledge might include general statements about the algorithm's best-case or worst-case behavior, and procedural knowledge might equate to an ability to provide the correct output of the algorithm for a given input.

Like EF Theory, cognitive constructivism situates knowledge squarely in the head of individuals. However, cognitive constructivism suggests that a qualitatively different form of knowledge arises out of effective learning environments—a form of knowledge not reducible to knowing a fact or skill. Papert (1980) cites the Logo programming environment as a prime example of a learning environment that supports the process of "getting to know an idea" (p. 136). In the Logo environment, students explore ideas by programming the Turtle—the equivalent of a pen with a heading—to draw. Papert argues that, in contrast to traditional forms of learning, in which learners merely memorize facts and practice skills, learners in the Logo environment "get to know the Turtle" by "exploring what a Turtle can

and cannot do" (p. 136). This activity, Papert contends, "is similar to the child's everyday activities, such as making mudpies and testing the limits of parental authority—all of which have a component of 'getting to know'" (p. 136).

Given that constructivist learning encompasses more than the acquisition of knowledge and skills, what are educators interested in evaluating the effectiveness of AV techology left with? Is it possible to measure the extent to which a learner has "gotten to know" an algorithm, in the cognitive constructivist sense? By closely examining a learner's progress over time, a skilled evaluator certainly could get a good sense of the learner's feel for an algorithm. However, it is important to point out two shortcomings of such an evaluation technique. First, it would most likely require significant amounts of time—more time than most evaluators are willing to spend. Second, such an evaluation might be criticized both because it is subjective, and because it is not replicable. Of course, one could allay some critics by making one's evaluation criteria public, and by using multiple evaluators and demonstrating a high level of agreement among them. However, these efforts would only serve to complicate the evaluation, adding to its already costly time requirements.

In the interest of practicality, then, it seems reasonable to consider less complicated, less time-intensive evaluation techniques. Here, the evaluation techniques adopted by EF Theory might serve as crude but acceptable alternatives. Indeed, notice that cognitive constructivism does not preclude the possibility that learners who have "gotten to know" an algorithm, in the constructivist sense, would perform well on conventional tests of facts and skills. As Papert (1980) puts it, learners who work in constructivist learning environments "certainly do discover facts, make propositional generalizations, and learn skills" (p. 136). Hence, if one hopes to perform a practical, replicable effectiveness evaluation of AV

technology with respect to the assumptions of cognitive constructivism, conventional tests of skills and knowledge may be a suboptimal, but viable alternative.

## Discussion: Algorithms as a Social Practice

The preceding section sketched out a learning theory that, like EF Theory, is squarely focused on individual knowledge. Learning, according to cognitive constructivism, involves learners' constructing their own knowledge by reorganizing and restructuring their existing knowledge. At first, concrete physical objects are involved in this knowledge construction process. Later on, learners build understandings out of more abstract conceptual objects.

The individualistic understandings emphasized by cognitive constructivism are certainly part of the competence that an algorithms student would ideally develop. On closer inspection of the activities and learning objectives of an algorithms course, however, I argue that "knowing" algorithms encompasses much more than cognitive skills. Rather, I suggest that the "schoolbook algorithms" taught in an undergraduate algorithms course is a fundamentally *social* endeavor; at its heart, it demands interaction with the social world of schooled algorithmaticians. Consequently, I contend that cognitive constructivism proves inadequate as a theoretical framework for interpreting the effectiveness of AV technology. What is required is a theoretical framework that views knowledge, and hence learning, at the level of *communities*.

To defend this position, let me first review the normative learning objectives of a typical undergraduate algorithms course. Upon completing an algorithms course, students would ideally be able to do such things as (a) write down a procedural description of a given

algorithm; (b) argue convincingly that an algorithm is correct; (c) analyze the efficiency of an algorithm; and, ultimately, (d) apply the appropriate algorithm to solve a given problem.

Notice that a student's level of competence in these skills cannot be objectively determined by the student's performance on a test. Rather, a student's level of competence is ultimately a matter of agreement among members of a community of expert "algorithmaticians" who are concerned with the conceptual foundations of algorithms. Consider, for example, how one might determine the "goodness" of a procedural description of an algorithm. In the context of a programming environment, a sufficient description is one that, when entered into a computer, actually compiles and executes as expected. In contrast, within the scope of an algorithms course, no programming environment is enlisted as the ultimate judge of the description. Instead, whether a given procedural description is sufficient depends largely on whether those who read or listen to it can understand it—that is, on whether the description communicates the algorithm in understandable terms. This, in turn, depends on whether the description makes use of an established language for describing algorithms—most often some sort of pseudocode.

Likewise, whether a given proof or efficiency analysis is convincing and correct depends largely on whether its audience is convinced and deems it correct; indeed, it has often been said that "proof is a social interaction." Once again, success depends not only on the way in which a student presents the proof or efficiency analysis, but also on the extent to which the proof or efficiency analysis appropriately makes use of established notation, conceptual tools, and techniques, including loop invariants, induction, and Big-O notation.

Finally, although it may not be obvious, whether a student appropriately applies an algorithm to solve a given problem is also largely a matter of social consensus. Indeed, applying an algorithm is not done in a vacuum, but rather within a social environment in which others must be convinced that the algorithm is the right one for the problem. And

without some sort of established conventions with respect to what "an appropriate algorithm" is, it would be impossible to determine what it would mean to choose the "appropriate" algorithm for a given problem.

In short, while the cognitive skills required to supply written answers to test problems are certainly important in an undergraduate algorithms course, I suggest that the competence that algorithms teachers would like students to acquire goes beyond an ability to answer written test problems adeptly. What teachers would really like their students to learn *is an ability to engage with competence in the practice of algorithms*. Here, the notions of "competence" and "practice of algorithms" have distinct meanings; let me briefly clarify them.

On the one hand, competence indicates a command of the conceptual tools, representations, language, and techniques that expert algorithmaticians use and recognize as appropriate. On the other hand, it indicates an ability to use those tools, representations, languages, and techniques in practice—that is, in a way that established algorithmaticians recognize and find convincing. By "practice of algorithms," I mean the typical activities in which those who are interested in the conceptual foundations of algorithms engage. Key among such activities is *algorithms discourse*—that is, conversations and dialog about algorithms, including how an algorithm works, how efficiently it runs, and why it is correct.

If one accepts the premise that an undergraduate algorithms course is really about training students to become competent members of a community of practice, one realizes the inadequacy of cognitive constructivism's purely individualistic view of knowledge and learning. In the case of an undergraduate algorithms course, an adequate theoretical framework must account for students' development as social members of a community that recognizes and values competence at such activities as algorithm explanation, analysis, and proof, all of which have a rich heritage of established techniques, tools, and language. Below,

I introduce a more recent version of constructivist theory that, because it views knowledge and learning at the level of the community, would appear to be a more appropriate theoretical framework for analyzing the effectiveness of algorithm visualization technology.

## Sociocultural Constructivism

Spurred by a "growing disillusionment with the individualistic focus" of cognitive theories of learning (Cobb, 1996, p. 34), sociocultural constructivism proposes a fundamental reinterpretation of the constructivist ideas discussed above.[11] Like cognitive constructivism, sociocultural constructivism views knowledge not as cognitively-instantiated representations of an objective reality, but rather as a constructed entity. However, whereas cognitive constructivism focuses on knowledge at the level of the *individual*, sociocultural constructivism focuses on knowledge at the level of the *community*. According to the former, individuals construct their own knowledge by assimilating their experiences into their existing knowledge. According to the latter, knowledge is socially constructed; it is the collaborative achievement of persons engaged in the practices of a community.

In shifting its epistemological focus from the individual to the community, sociocultural constructivism understandably stresses a markedly different set of assumptions regarding learning. Most significantly, for sociocultural constructivism, learning only makes sense within the context of participating in the ongoing activities of a particular community

---

[11]Sociocultural constructivism has evolved out of the work of a number of social scientists working within different disciplines, including social psychology, anthropology, sociology, and educational research. Not surprisingly, these social scientists have had different agendas, have used different terminology, and have emphasized different aspects of the position. The account of the theory I offer here draws primarily on the influential work of anthropologist Jean Lave and her Ph.D. students (Lave & Wenger, 1991; see also Lave, 1988, Lave, 1993; Lave, 1997; Wenger, 1998). Note that Lave and colleagues label their position *situated learning theory*, not sociocultural constructivism.

of practice (Lave & Wenger, 1991); skill and knowledge do not exist independently of such a context. Moreover, sociocultural constructivism views learning in terms of changing participation; to learn is to participate, in increasingly competent ways, within a community. Here, the concept of *identity*—the way one views oneself, and is viewed by others with respect to the community—plays an important role; one's identity undergoes significant changes as one's level of participation changes within the community. Finally, the sociocultural constructivist position emphasizes that *access* and *opportunities for participation* are essential to the learning process. Without access to the practices, resources, and members of the community, and without opportunities to participate in the practices of the community, learning would be impossible.

Below, I elaborate further on sociocultural constructivism. I begin by relating an alternative version of the Allegory of Musica introduced in Chapter II. The intention of this alternative allegory is not only to highlight the key assumptions of sociocultural constructivism, but also to contrast it with EF Theory. I then move to a discussion of the implications of the position for design, pedagogy, and evaluation. As I shall illustrate, sociocultural constructivism does not contradict cognitive constructivism. Rather, it reinterprets the benefits of so-called active learning, placing greater emphasis on the importance of authentic activities and collaboration.

## The Allegory of Musiphonia

Recall that the Allegory of Musica (see Chapter II) tells of a singing mammal, the Musibeest, that once roamed the land of Musica, and of the Musican school system's commitment to teaching students to imitate the Musibeest's song as accurately as possible. In the neighboring land of Musiphonia, citizens also enjoy the songs of the Musibeest.

However, their Musican counterparts, in a historic invasion, captured and took away all of the Musibeests living in Musiphonia. Shortly after the Musican invasion, a few Musiphonians set about the task of preserving the heritage of the Musibeest song. With the help of a few surviving Musibeest recordings, they developed a musical brass ensemble instrument called the Musiphone. Its inventors dedicated themselves to organizing Musiphone ensembles, whose charter was to celebrate and share the song of the Musibeest.

Unlike the Musicans, the Musiphonians were not concerned with the accuracy of their Musibeest reproductions. (Even if they had been interested in accurately reproducing Musibeest sounds, they no longer had in their midst Musibeests against whom to check their reproductions.) Instead, the Musiphonians turned their interest to building cohesive, tightly-knit Musiphone ensembles to experience—and, indeed, to define—what it means to know Musibeest music.

Within the Musiphone ensembles, a system of educating newcomers gradually evolved that was radically different from that of the Musicans. Musiphonians who decided they wanted to dedicate themselves to the music of the Musibeest were taken in by Musiphone ensemble masters. When they joined the ensemble, those newcomers were given simplified versions of the Musiphone and small, simple parts in the concerts. As they gradually gained experience, they were given increasingly sophisticated versions of the Musiphone, and they were called on to play increasingly difficult parts in concerts. In addition, they might be asked to help the Musiphone masters out with other administrative responsibilities entailed in Musiphone ensemble membership—for example, booking concerts, maintaining instruments, and scheduling consultations with interested newcomers. Eventually, a newcomer would gain so much responsibility in the ensemble that her role would be hard to distinguish from that of the ensemble master. Indeed, she would gradually come to be recognized as a master in her own right.

## Key Assumptions

### Knowledge and Learning within a Community

As the Allegory of Musiphonia indicates, sociocultural constructivism differs markedly from both EF and cognitive constructivist theory with respect to the status of knowledge and the process of learning. Whereas the EF and cognitive constructivist perspectives situate knowledge in the head of the individual, the sociocultural constructivist view reconceptualizes knowledge as the collaborative achievement of a community. As Lave and Wenger (1991) note, "[a] community of practice is an intrinsic condition for the existence of knowledge" (p. 98).

A contrast between the Allegory of Musica (Chapter II) and the Allegory of Musiphonia (above) serves to elucidate this view. For both the Musicans and the Musiphonians, knowledge is intimately entwined in the songs of the Musibeest. The relationship between knowledge and Musibeest songs, however, is markedly different in the two allegories. In Musica, knowledge fundamentally entails replicating a coveted absolute: the song of the Musibeest. The closer the match with that absolute (a close match can be measured objectively), the truer the knowledge. In Musiphonia, on the other hand, knowledge—that is, the song of the Musibeest—does not exist except in relation to people engaging in the practices of a community; it is the collaborative achievement of Musiphonian ensembles making music together.

This view of knowledge implies a fundamentally different position with respect to what it means to learn. As we have seen, both the EF and cognitive constructivist views focus on learning at the level of the *individual*. These views hold that individual knowledge

is somehow transformed through the learning process, and they aim to explain just how such changes in individual knowledge come about. By contrast, sociocultural constructivism interprets learning at the level of the *community*. Instead of asking "What knowledge does one gain through learning activity *x*," sociocultural constructivism asks "What form of participation in community *y* is enabled through learning activity *x*?" As Lave and Wenger (1991) emphasize, "participation in the cultural practice. . .[of a community]. . .is an epistemological principle of learning" (p. 98).

In the Allegory of Musiphonia, for example, the focus is not on individual learning, as evidenced by an ability to replicate Musibeest songs with increasingly high accuracy. Rather, the allegory portrays learning in terms of participating more fully within a Musiphonian ensemble. Fuller participation, on this view, entails gaining the kinds of competence necessary to take on the more central practices of the community, such as playing more important parts in concerts with more complicated instruments, and arranging meetings with newcomers.

## Learning Is Changing Participation and Identity

Whereas EF and cognitive constructivist theories focus on changes in the mind of the learner, sociocultural constructivist theory focuses on changes in the way in which the learner *participates* in the practices of a community. According to this view, a community implicitly defines a participation structure, which encompasses various ways of participating in the community and thereby contributing to its reproduction. When newcomers enter a community, they participate at its periphery; their role in contributing to the ongoing practices of the community are minimal. According to Lave and Wenger (1991), such peripheral participation, which is legitimized by the community, defines the learning

process. It is only through the process of *legitimate peripheral participation* that newcomers are able to learn—that is, to participate in more "expert" ways within a community.

For instance, in the Allegory of Musiphonia, newcomers start out by playing small parts on simple versions of the Musiphone. In this early stage in their Musiphone Ensemble career, they are participating on the periphery. Gradually, as they participate for a longer time, they may be given the opportunity to participate in more central ways. For example, they may play more important parts in concerts, or they may take on more administrative responsibility. In so doing, they move toward fuller participation in the community. If they stay with a Musiphone Ensemble long enough, they may gain a level of participation that is indistinguishable from that of a Musiphone master.

A key aspect of a community member's advancement toward fuller participation is the member's *identity*: how the member views herself, and is viewed by others, within and with respect to the community. On the sociocultural constructivist view, one's status within a community depends not only one's level of competence and participatory role in the practices of the community, but also on one's own perceptions of one's place in the community, as well as those of others. As Lave and Wenger (1991) put it,

> Activities, tasks, functions and understandings do not exist in isolation; they are part of broader systems of relations in which they have meaning. The person is defined by as well as defines these relations. Learning thus implies becoming a different person with respect to the possibilities enabled by these systems of relations. To ignore this aspect of learning is to overlook the fact that learning involves the construction of identities. (p. 53)

The Allegory of Musiphonia can be used to illustrate the role of identity in learning. At first, an ensemble newcomer participates in ways that indicate she is a newcomer. She plays simple parts on simple instruments, and, for the most part, only observes fuller members as they take care of the business of the ensemble. She regards herself as a beginner in the ensemble, and others also recognize her as such. As she moves through the ranks of the ensemble, however, she constructs a new identity for herself. As she is given the

opportunity to play more important parts in concerts, and as she is granted more administrative responsibilities, she comes to see herself as a more central player in the ensemble. Likewise, others regard her as such, and, as a result, begin to treat her differently. For instance, they might call her by different names or greet her differently. The most significant consequence of her change in identity, however, is her change in level of participation: She takes on a more central role in the community, and does so with the support and blessing of other community members.

Access Is Crucial

As discussed above, the sociocultural constructivist view maintains that learning can only take place through participation in the authentic activities of a community of practice. An important implication of this position is that *access* to the community is absolutely essential to the learning process. As Lave and Wenger (1991) explain, through access to the community, learners are able to become more central members of the community:

> From a broadly peripheral perspective, [learners] gradually assemble a general idea of what constitutes the practice of community. This uneven sketch of the enterprise (available if there is legitimate access) might include who is involved; what they do; what everyday life is like; how masters talk, walk, work, and generally conduct their lives; how people who are not part of the community of practice interact with it; what other learners are doing; and what learners need to learn to become full practitioners. It includes an increasing understanding of how, when, and what they enjoy, dislike, respect, and admire. In particular, if offers exemplars (which are grounds and motivation for learning activity), including masters, finished products, and more advanced [learners] in the process of becoming full practitioners. (p. 95)

This quote suggests that learners require access to a broad range *of community resources*, including the ongoing activities of the community, central members of the community, other learners, and the tools, artifacts, and technology of the community. In Musiphonian Ensembles, for example, newcomers require access to the ensemble leader, other ensemble members, sheet music, Musiphones, performances, and rehearsals.

In addition to community resources, sociocultural constructivism emphasizes that learners require access to *participation* in the authentic activities of the community—both peripheral forms, and more central forms. For instance, in the Musiphonian ensembles, newcomers are initially allowed to participate in the ensemble peripherally; they play simplified instruments and small parts. Later on, as they mature as ensemble members, they gain access to more central forms of participation. For example, they play more important parts in concerts, and help to recruit newcomers. In sum, access to resources and participation in a community is crucial if learners are to learn—that is, if they are to gain fuller membership in the community.

## Implications for Pedagogy

Sociocultural constructivism, as articulated by the assumptions just discussed, has important implications for how algorithms instructors ought to use AV as a pedagogical aid. As we have seen, sociocultural constructivism views learning in terms of increasingly central participation within a community of practice. I call the community that is in the process of reproducing itself through an undergraduate algorithms course the "Community of Schooled Algorithmaticians" (COSA). This community consists of people who "practice" algorithms with a shared sense of such things as (a) what is important and interesting about algorithms (e.g., procedural behavior, efficiency, correctness); (b) how to communicate about algorithms with others (e.g., comparisons of naïve and efficient algorithms, pseudocode, graphical notations, correctness proofs, Big-O analyses); and (c) how to do algorithmic problem solving.

As we have seen, sociocultural constructivism assumes that learning algorithms fundamentally entails participating more centrally in the COSA, and that learners can only advance toward more central participation if they have access to the community. With

respect the pedagogical use of AV technology, these assumptions have a clear message: namely, that algorithms instructors would do well to set up situations in which students have access to (a) the *resources* of the COSA (e.g., ongoing activities, old-timers, learners, AV technology), and (b) *opportunities for participation* in the authentic activities of the COSA. The important question for instructors thus becomes, "How might AV technology be used to provide opportunities for participation in the authentic activities of the COSA?"

I argue that AV technology, whether high-tech graphical simulations or low-tech sketches, provides access to at least three key forms of COSA participation:

1. meaningfully interpreting visual representations of algorithms (*AV reading*);
2. constructing visual representations of algorithms (*AV writing*); and
3. presenting visual representations of algorithms to, and discussing them with, members of the community (*AV presentation/discussion*).

Notice that these three activities are listed roughly in order of increasing centrality. Whereas newcomers in the community might be expected to read an algorithms text and meaningfully interpret the graphical representations contained therein, newcomers are seldom expected to construct their own visual representations of algorithms, much less to engage in presentations and discussions involving such representations. In other words, AV reading is a peripheral form of participation in the community; AV writing, and AV presentation/discussion are more central. These layers of AV technology-mediated participation, which define an important part of the COSA participation framework, are illustrated in Figure 14.

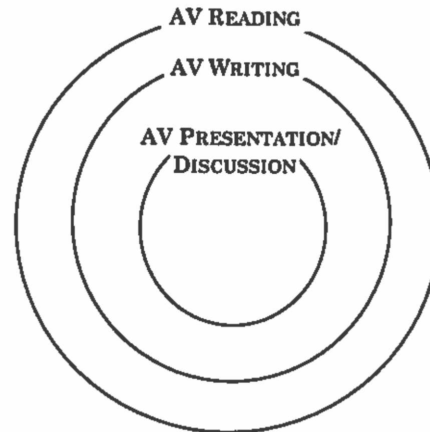



Figure 14. Layers of AV Technology-Mediated Participation

As I illustrated in Chapter II, the traditional pedagogical approach to using AV technology has been to have students view and interact with graphical representations of algorithms. On the sociocultural constructivist view, this approach grants students access to the activity of AV reading. However, if it is true, as sociocultural constructivism assumes, that access to *increasingly central* participation is key to learning, then algorithms instructors would do well to provide students with access to the more central forms of AV technology-mediated participation: AV writing and AV presentation/discussion.[12] Granting such access would involve setting up situations in which students construct their own AVs, and then present them to, and discuss them with, a group of COSA members—both old-timers and newcomers.

---

[12]As Lave and Wenger (1991) put it, "the understanding to be gained from engagement with technology can be extremely varied, depending on the form of participation enabled by its use" (p. 101).

## Implications for Technology Design

As discussed in Chapter II, EF Theory holds that the key to successful learning is a close denotational match between an expert's mental model and its graphical representation in an AV. As I argued, this view implies that AV technology must support high fidelity representations, which enable an algorithms expert to portray an algorithm with a high degree of accuracy. Extant AV technology, as I illustrated, has taken this implication of EF Theory to heart through its consistent support of input-general, high-typeset fidelity AVs.

In contrast, sociocultural constructivism asserts that AV technology's pedagogical value lies in its ability to provide access to increasingly central participation in the COSA. Clearly, the implications of this position for AV technology design differ markedly from those of EF Theory. Indeed, sociocultural constructivism, like cognitive constructivism, rejects the importance of a close denotational match between an expert's mental model and an AV. Instead, sociocultural constructivism implies that AV technology designers ought to concentrate on building AV technology that supports both *participation* and *communication*.

Sociocultural constructivism suggests that, to support increasingly central forms of participation in the COSA, AV technologists must jettison the traditional dyadic user model discussed in Chapter II, as this dyadic model fosters the student-instructor dichotomy, and denies students access to more expert forms of participation. But what user model should be used in its place? Like cognitive constructivism, sociocultural constructivism suggests that AV technology should enable students to participate in AV construction—that is, it should provide an environment in which students can construct their own AVs. However, as we have seen, sociocultural constructivism goes beyond cognitive constructivism in asserting that AV technology must also enable students to participate in the most central AV technology-mediated activities: AV presentation and discussion.

According to sociocultural constructivism, it is in the activities of AV presentation and discussion that AV technology holds promise in facilitating effective communication about algorithms. In this capacity, sociocultural constructivism does not see AVs as merely "transferring" knowledge from the AV presenter to the audience, as EF Theory holds. Rather, according to the sociocultural constructivist view, AV technology serves to *mediate* interaction between students and instructors. In the words of Roschelle (1990), AV technology serves to manage the inherent "uncertainty of meaning in conversations" by functioning as "a common ground for shared activity and talk among learners and experts" (p. 3).[13]

The Allegories of Musica and Musiphonia well juxtapose the communicative role of AV technology according to EF Theory, on the one hand, and according to sociocultural constructivism, on the other. In Musica, AV technology is analogous to a tape recorder; it conveys with precise accuracy the song of the Musibeest, which symbolizes absolute knowledge. In Musiphonia, on the other hand, AV technology is analogous to the Musiphone; it enables members of Musiphonian ensembles to negotiate shared meaning—the socially agreed-upon song of the Musibeest—through their concerts. Clearly, to play the role of a Musiphone, AV technology must support conversations about algorithms. Just what specific design features this implies is a question that only an empirical study can answer; I take the question up in the next chapter.

---

[13]Roschelle (1990) uses the term *symbolic mediation* to refer to the use of visual representations in this way—that is, as a mediator of conversations.

## Implications for Effectiveness Evaluation

According to sociocultural constructivism, effective pedagogical exercises involving AV technology provide access to increasingly central forms of participation in the COSA, including AV construction and AV presentation/discussion. Thus, one immediate measure of effectiveness suggested by sociocultural constructivism is *activity relevance*: to what extent does a pedagogical activity involving AV technology involve learners in the authentic activities of the COSA? The more relevant and authentic is a learner's experience with respect to the COSA, the better.

The sociocultural constructivist position further suggests that, through participating in increasingly central ways, learners gradually become fuller members of the COSA. Yet, how should one measure something as ostensibly vague as membership in a community? Recall that one's level of membership, on the sociocultural constructivist view, is intimately connected to one's *level of participation* and *identity*. As discussed above, the roles, responsibilities, and tasks that one takes on within a community fit into a participation framework that resembles an onion: Some of these roles, responsibilities, and tasks are on the periphery (the skin of the onion); others are more central (near the core of the onion). Thus, the roles, responsibilities, and tasks that one takes on within a community provide a rough indication of one's level of participation, and hence the extent to which one has gained full membership. By conducting ethnographic fieldwork within a community of practice, one can observe longitudinal changes in people's levels of participation within the community.

As discussed above, a key component of level of membership is *identity*: how one perceives oneself, and is perceived as others within and respect to the community. Like level of participation, identity is difficult to measure in a quantitative sense. However, just as it may be used to assess one's level of participation, ethnographic fieldwork can be used to gain

a qualitative sense of the way in which one's identity with respect to a particular community of practice changes over time.

## Summary and Research Questions

Inspired by the empirical trend, noted in the last chapter, that suggests that increased student involvement in the visualization process leads to higher learning outcomes, this chapter has proposed constructivism as an alternative, more appropriate guiding theory for the pedagogical use of AV technology. Cognitive constructivism, the first brand of constructivism I considered, views knowledge as a highly subjective, individually-constructed entity. In contrast to EF Theory, the cognitive constructivist position would predict the empirical finding that active viewer involvement is important. In particular, cognitive constructivism implies that, since learners must actively construct their own understandings, they will learn best by constructing their own visual representations of algorithms.

This chapter has suggested, however, that understanding algorithms entails far more than an ability to recount accurately procedural behavior, that understanding algorithms fundamentally involves an ability to engage in the *practice of algorithms*. Since that practice is inherently social, I have argued that cognitive constructivism, with its narrow focus on individual knowledge, proves inadequate as a guiding theory of effectiveness. As a consequence, I turned to sociocultural constructivism, a more recent version of the constructivist position that situates knowledge within communities of practice, and recasts learning in terms of changing levels of participation in such communities. I argued that sociocultural constructivism, with its broader analytical scope, proves more appropriate as a

guiding theory, and I outlined its implications for the pedagogical use, design, and evaluation of AV technology.

While many of its implications were similar to those of cognitive constructivism, two key differences did arise: first, that sociocultural constructivism emphasizes the importance of providing students with opportunities to participate in the social, AV technology-mediated activities of AV presentation and discussion; and second, that sociocultural constructivism stresses the importance of designing AV artifacts as *mediational resources* for such presentations and discussions.

As we have seen, sociocultural constructivism offers plenty of guidance with respect to how to design, implement, and gauge the benefits of AV technology-based pedagogical exercises. Yet, until we actually try out its ideas within the context of an actual algorithms course, we cannot know whether sociocultural constructivism is appropriate as a guiding theory of effectiveness for AV technology. As we have seen, in order to observe the outcomes of the sociocultural constructivist ideas, one must view the algorithms course in which they are implemented as a distinct community. And one way to proceed to do this is to conduct an ethnographic field study. In the next chapter, I report on a series of exploratory ethnographic studies that I conducted in two different algorithms courses that incorporated AV technology using a sociocultural constructivist approach.

# CHAPTER IV

# ETHNOGRAPHIC STUDIES

> The study of ethnology—so often mistaken by its very votaries for an idle hunting after curios, for a ramble among the savage and fantastic shapes of "barbarous customs and crude superstitions"—might become one of the most deeply philosophic, enlightening and elevating disciplines of scientific research. Alas! The time is short for ethnology, and will this truth of its real meaning and importance dawn before it is too late?
> (Malinowski, 1922, p. 518)

Through a method variously referred to as ethnography or ethnology, anthropologists have established a rich tradition of exploring cultures through in situ field studies (see, e.g., Estroff, 1981; Malinowski, 1922; Mead, 1928; Spradley, 1970). Through a document known as an *ethnography*, the ethnographer aims to describe a culture from the perspective of an insider. To do so, the ethnographer typically lives within the community for a period of time, making use of any of several established ethnographic field techniques, including (a) *participant observation*—experiencing first-hand the authentic practices of the community, either actively as a participant, or more passively as an observer; (b) *fieldnotes*—maintaining a written record of one's experiences in the field; (c) *interviews*—talking with community members about their views and experiences, either with a set list of questions, an open-ended list of questions, or spontaneously; (d) *questionnaires*—asking community members a set of questions through a written instrument; (e) *artifact collection*—gathering and analyzing the artifacts of the community; (f) *diary collection*—asking community members to keep a diary of their experiences, thoughts and activities; and (g) *audio- and videotaping*—recording the events of a community on audio- or videotape for later analysis.

Far from being the exclusive tools of ethnographers, 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; Springmeyer, 1992). 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.

As I argued in the previous chapter, sociocultural constructivism views learning at the level of the *community*. The value of a learning exercise, according to sociocultural constructivism, rests in its ability to provide access to, and ultimately to facilitate, increasingly central participation within a community. AV technology, on this view, is pedagogically valuable insofar as it provides students with access to increasingly expert forms of participation, and insofar as it serves to mediate meaningful learner-learner and learner-expert conversations about algorithms. Given this community-based view of learning, ethnographic field techniques would appear well-suited to assessing the impact of AV technology-based learning exercises from a sociocultural constructivist perspective. Indeed, ethnographic field techniques hold promise in providing a rich understanding of the ways in which AV technology-based learning exercises both facilitate participation within the community being reproduced through an algorithms course, and support communication about algorithms.

In this chapter, I present the key findings of a series of two ethnographic field studies that explored the effects of implementing, within an algorithms course, AV technology-based learning exercises rooted in sociocultural constructivism. Each of these studies considered a separate offering of a third-year undergraduate algorithms course taught by the same instructor during consecutive, ten-week terms at the University of Oregon. While the scope

of these studies initially was broader,[14] this chapter focuses on reporting and discussing those study findings that address three specific research questions raised by the sociocultural constructivist position:

1. Do AV construction and presentation exercises engage students in activities that are relevant to the Community of Schooled Algorithms (COSA)?
2. How should AV technology be designed so as to mediate presentations of, and conversations about, algorithms?
3. Do AV construction and presentation exercises help students to gain fuller membership in the COSA?

The chapter begins 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; who the informants in the study were; and what field techniques I employed in the fieldwork. Next, I present key observations, which relate to students' activities and experiences both in constructing and presenting their AVs. Finally, I discuss the observations vis-à-vis the three key research questions listed above. As I shall illustrate, the observations made during these studies provide crucial insights into the three key research questions listed above.

The most significant of these insights is that AV construction assignments,[15] when supported by high epistemic fidelity AV technology, can actually distract students from participating in COSA-relevant activities. When supported by low epistemic fidelity technology, however, AV construction and presentation/discussion appear to enable students

---

[14]Consult Appendices A and B for more detailed accounts of these ethnographic studies.

[15]Note that I shall use "AV construction assignments," "visualization construction assignments," and "animation construction assignments" interchangeably in this chapter.

to participate more extensively in COSA-relevant activities, thus contributing to students' gaining more central membership in the COSA.

## 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.

As indicated by the sample syllabus included in Appendix C, the CIS 315 course in which I conducted my fieldwork adopted a standard text (Cormen, Leiserson, & Rivest, 1990). The course 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 (or final programming project), and various algorithm visualization assignments.

In the remainder of this section, I briefly describe the students and instructor who participated in the two courses I observed, the animation assignments that provided a backdrop for my fieldwork, and the field techniques I employed to gather data.

## 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,[16] 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 fellow graduate student of mine 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.

## Algorithm Visualization Assignments and Supporting Technology

Based on the sociocultural constructivist recommendations outlined in the previous chapter, the visualization assignments that Professor Lane and I devised for the course included both a construction component, and a presentation/discussion component.

However, these components differed substantially in each of the course offerings that I studied.[17]

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.[18] 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 high epistemic fidelity, multiple-window, color animations.

---

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

[17]I shall refer to the first academic quarter of my fieldwork as "Study I," and the second academic quarter of my fieldwork as "Study II."

[18]Consult Appendix C for copies of the actual assignments statements given to students.

The Samba interpreter takes as input a text file containing a series of Samba commands (one per line), and generates a corresponding animation. Figure 15 presents a fragment from a sample Samba script, along with a snapshot of the animation generated by the script.

Once a Samba animation has been programmed, it can be viewed using the tape recorder-style interface illustrated in Figure 16. 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 15(b)] allow one to zoom in and out of the view, and to pan around within the view.

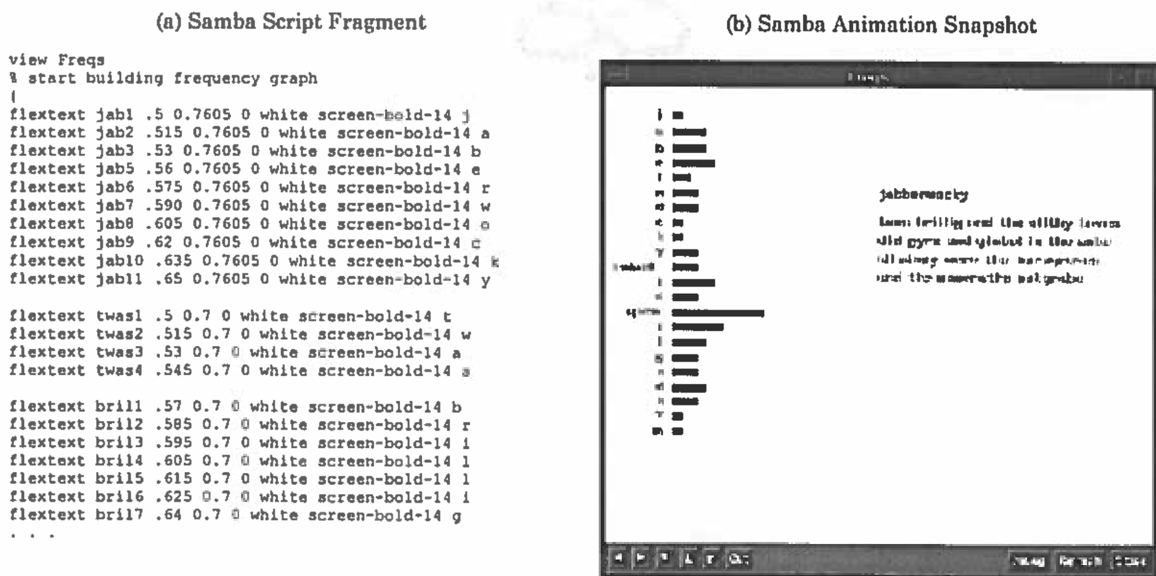



Figure 15. Sample Samba Script Fragment and Animation Snapshot

Figure 16. The Polka Control Panel

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 low epistemic fidelity "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.

## 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 my dual-role of student observer and teaching assistant, I employed at least seven different ethnographic field techniques.[19] 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* (see Appendix C) 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.

---

[19]Just what makes my studies "ethnographic" is an important consideration. In this chapter, I have space only for a brief discussion of the ethnographic field techniques I used. Consult Appendix A for a more thorough treatment of this topic.

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.

## 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 17 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 18 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 represeutions*; they depicted their target algorithms using the canonical, purely geometric representations that appear in the course textbook.

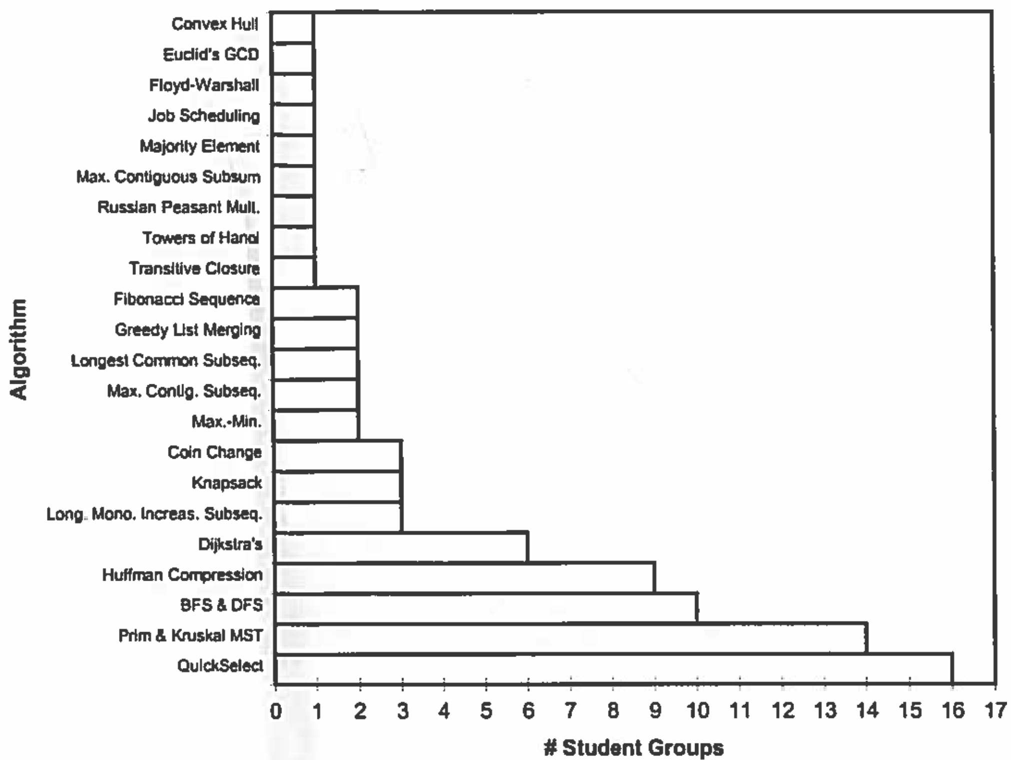



Figure 17. Algorithm Themes Animated by Student Groups

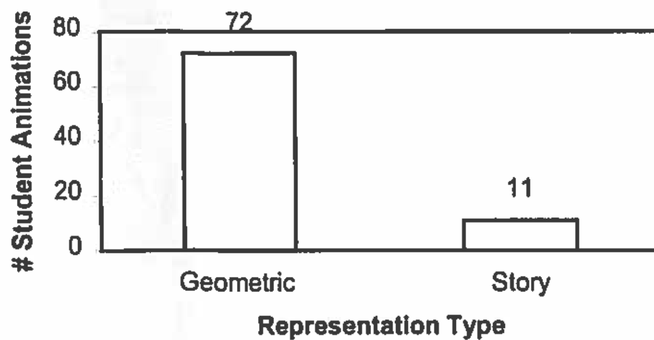



Figure 18. Number of Geometric and Story-Based Animations

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, Leiserson, & Rivest, 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[20], which portrays the algorithm for solving the longest common subsequence problem[21]:

> The world floods again, and Donald Knuth[22] 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

---

[20]See Appendix B for a detailed account of this animation and the discussions that it stimulated.

[21]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, 1990, pp. 314–319).

[22]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).

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.

## Animation Construction

Based on the diary data I collected in Study II, Figure 19[23] 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.[24] 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.

---

[23]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.

[24]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.

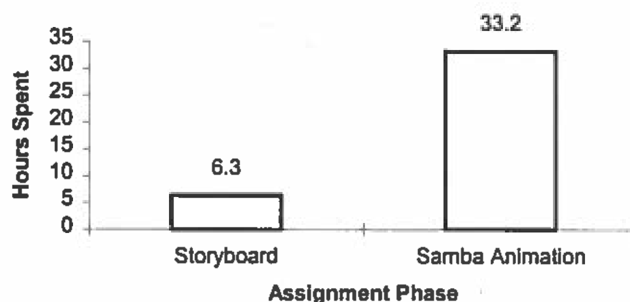



Figure 19. Time Spent on Storyboards and Samba Animations

Storyboard Construction Activities

Based on an analysis of student diaries, Figure 20 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) heeded the advice of the handout "Guidelines for Developing a Storyboard Presentation" (see Appendix C)[25] by preparing 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).

---

[25]I based these guidelines on my experiences with using the visualization storyboarding technique in prior research (Douglas, Hundhausen, & McKeown, 1995, 1996).

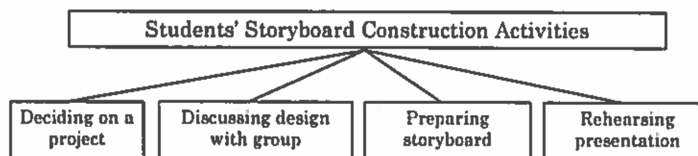



Figure 20. Students' Storyboard Construction Activities

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

Samba Construction Activities

Derived from interview and questionnaire data collected in Study I, Figure 21 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 (see Appendix C), 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 21: implementing a "driver" algorithm.[26] Unfortunately, Professor Lane did not supply students with 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 admittedly scant questionnaire data, algorithm programming activities, on average, accounted for the largest portion of the overall time spent on the second Study I assignment.

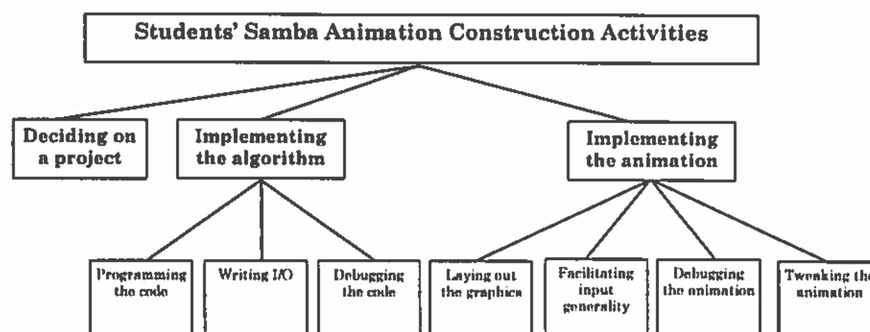



Figure 21. Students' Samba Animation Construction Activities

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; the lower left-hand corner of the screen is 0,0, and the upper right-hand corner of the screen is 1,1. Since they did not

---

[26]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).

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 21. 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.

## 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.

### 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.[27]

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?").

---

[27]Note that these observations concur with those of prior research into storyboard presentations (Chaabouni, 199; Douglas, Hundhausen, & McKeown, 1995, 1996;).

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. On the other hand, other suggestions were collaborative achievements; they arose out of discussions in which Professor Lane and student presenters considered alternative designs.

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 4.

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 Appendix B for a vivid example.

Table 4. 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 $n^{th}$ 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? |

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.[28]

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

## 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 research questions posed at the beginning of this chapter. Plainly, great care must be taken in any attempt to generalize the findings I discuss beyond the particular algorithms course I studied.[29]

### Activity Relevance

The observations presented above motivate two insights into the relevance of AV construction and presentation exercises with respect to the COSA. The first of these is that

---

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

[29]Aside from having limited generalizability, my ethnographic fieldwork was influenced by the biases I brought to the field; see Appendix A for a discussion of these biases.

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 COSA. As Professor Lane himself confided, "These are useful exercises, but not in this class."

By contrast, in constructing input-specific, low-fidelity 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 COSA—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 burn 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: Low-fidelity 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 low-fidelity 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). On the other hand, 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, frontwards 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.

## Designing for Conversations about Algorithms

As discussed in Chapter II, ever since Brown's (1988) pioneering work on BALSA, 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 viewing. Indeed, Brown's (1988) tape recorder-style interface for interacting with AVs (see Chapter II) has become the taken-for-granted standard in nearly all subsequent AV technology. However, in light of the observations of the student presentation sessions described above, 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, the observations suggest that *conversation-supporting AV technology* requires three user interface features not supported by extant AV technology.

First, as the 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 additional 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.

AV Construction and Presentation as Expert Forms of COSA Participation

According to sociocultural 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, and hence students' fuller participation in the community. 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 COSA. 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 COSA. 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.

## Summary

This chapter has presented a series of ethnographic observations that explored the value of AV construction and presentation exercises in an actual undergraduate algorithms course. The observations I made in these studies appear to make a strong case for the plausibility of the sociocultural constructivist view with respect to the value of technology as a learning aid. As might have been predicted by sociocultural constructivism, Professor Lane's and my initial approach to implementing construction and presentation exercises was naïve. We failed to recognize that requiring students to construct and present high epistemic fidelity AVs would prove so distracting. By shifting from Samba to storyboards, we discovered that students not only spent significantly less time on the construction part of the exercises, but that the activities in which they engaged were more relevant.

Moreover, the observations I made during the presentation sessions underscored the role of AV technology in mediating meaningful conversations about algorithms. As we have seen, the shift in the ontological status of AV technology advocated by sociocultural constructivism—from *knowledge conveyors* to *conversation mediators*—has important

implications for the design of the technology. Finally, my observations highlighted AV technology's role, within AV construction and presentation exercises, in providing students with access to expert forms of COSA participation. As suggested by the sociocultural constructivist position, access to such participation transforms students' competence and identity, thus enabling them to gain more central membership in the COSA.

# CHAPTER V

## A FRAMEWORK OF CAUSE AND EFFECT

Science may be described as the art of systematic over-simplification.
K. Popper, as quoted in *Observer* (London, 1 Aug. 1982)

The overriding purpose of the ethnographic studies described in the previous chapter was to bring into sharper focus the costs and benefits of AV technology in general, and of the sociocultural constructivist pedagogical approach in particular. As we have seen, on a practical level, these studies offer several recommendations for how best to implement AV construction assignments in an undergraduate classroom. For example, the assignments should not include an input generality requirement; they should not necessarily require students to implement their AVs on a computer; and they should provide opportunities for students to present paper-and-pencil storyboards to their instructor and peers for feedback and discussion.

On a more theoretical level, these studies lay a solid empirical foundation on which to develop a specific set of hypotheses with respect to how and why algorithm visualization artifacts might be effective pedagogical aids. In an effort to formulate such a set of hypotheses, I ask three framing questions in this chapter:

1. What are the central factors that influence (cause) the effectiveness of pedagogical exercises involving AV artifacts?
2. What are the precise effects of those factors? In other words, what measures can be used to gauge the benefits of the factors?
3. What are the linkages between factors and measures? In other words, what factors or combination of factors cause what effects or combination of effects?

By answering these questions in light of the ethnographic studies presented in the previous chapter, this chapter develops a framework of cause and effect for AV effectiveness. The framework expands on and refines the repertoire of factors, measures, and hypotheses that have been explored by past empirical research. As discussed in Chapter II, past experiments have focused squarely on causal factors suggested by various versions of EF Theory, including Strong EF, and the Individual Differences, Dual-Coding, and Learner Involvement versions of Weak EF. Likewise, past experiments have explored a narrow range of knowledge transfer measures—primarily tests of procedural and conceptual understanding. In contrast, while still hypothesizing about EF Theory's notion of understanding and recall, the framework presented here includes additional hypotheses that EF Theory simply could not predict. These additional hypotheses tailor the sociocultural constructivist position so that it applies to the particulars of algorithms learning. The result is a richer characterization of AV effectiveness—one that, in broadening its focus to the community of practice, goes beyond the confines of individual cognition to which EF Theory limits itself.

This chapter begins by describing, in greater detail, the five causal factors and four measures included in the framework. I then discuss the centerpiece of the framework: a series of four hypotheses that link cause and effect. Finally, I summarize the framework, and introduce two important research questions it raises. These research questions motivate the two alternative research directions—a series of experiments, and a prototype AV system—pursued in Chapters VI and VII.

## AV Effectiveness Factors

In the ethnographic studies reported in the previous chapter, six factors stood out as strongly influencing the effectiveness of AV-based pedagogical exercises. These factors appear as leaf nodes in the taxonomy presented in Figure 22. The subsections that follow elaborate on each of the factors in the taxonomy, drawing from both the findings of the ethnographic fieldwork, and from related research.

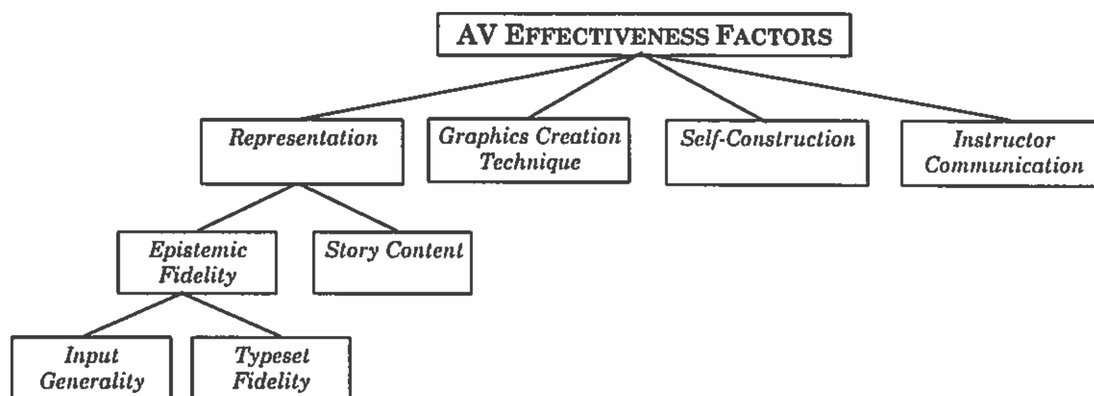



Figure 22. A Taxonomy of AV Effectiveness Factors

### Representation Factors

The *representation* factors relate to the nature of visual representations used within AV-based pedagogical exercises; this includes their functionality, appearance, and content. The following subsections describe *input generality*, *typeset fidelity*, and *story content*, three factors relating to AV representation.

Input Generality and Typset Fidelity

As discussed in Chapter II, the *epistemic fidelity* of an AV is the extent to which it provides a faithful rendition of the algorithm it is intended to illustrate in terms of an expert's mental model. Two factors influence the level of an AV's epistemic fidelity:

1. *Input generality*—Does the AV depict the algorithm for general input?
2. *Typeset fidelity*—Does the AV have the polished, computer-generated look typical of AVs that appear in textbooks?

As discussed in Chapter II, both of these characteristics are direct consequences of *direct generation*, the AV-creation technique that nearly all AV technology embraces, including the AV technology (Samba; Stasko, 1997) used in the ethnographic studies.

Recall that a key observation from the ethnographic studies was that students not only spent inordinate amounts of time on the animation assignments, but also engaged in activities that were irrelevant and distracting with respect to the focus of the algorithms course in which they were enrolled. The reason for this, as I argued in the last chapter, was that students' AVs were required to have both *input generality* and high *typeset fidelity*. Because students had to worry about getting their animations to work for general input, and because they spent a lot of time tweaking their AVs so that they had high typeset fidelity, they ended up spending a lot of time on programming tasks that were unrelated to the actual algorithm they were trying to illustrate.

By contrast, when we eliminated input generality and typeset fidelity requirements in Study II, we saw students spend not only significantly less time overall on the assignment, but also a higher percentage of their overall time on activities that required them to focus on the algorithms they were animating. This finding suggests that, in the interest of

maximizing students' time on relevant, potentially beneficial activities, student-constructed AV assignments would do well not to require input generality and typeset fidelity.

Story Content

As reported in Chapter IV, a small minority of student groups (13%) elected to construct AVs based on stories or scenarios, in which real or fictitious human beings are engaged in some problem-solving venture. It turned out that AVs with story content had two main advantages over AVs that were not based on stories. First, students who constructed stories appeared more motivated, enthusiastic, and engaged. They appeared to take great pleasure in coming up with original stories, and in building AVs around those stories. Second, student presentations of story-based AVs clearly captured the attention of their audience, and led to more lively discussions, than did AVs that were not based on stories.

In addition to these observed benefits of story-based AVs, one can cite a more speculative advantage of story-based AVs. Several studies in the psychological literature (see, e.g., Bower & Clark, 1969; Hill, Allen, & McWhorter, 1991) establish an empirical basis for the value of stories as *mnemonic devices*—that is, techniques that help to improve one's memory of items to be learned. In these studies, the construction of personally-meaningful stories containing words to be learned consistently led to improved recall not only of the words themselves, but also of the order of the words. Researchers account for this improvement by hypothesizing that personally-meaningful stories create a highly-interconnected network of vivid images that cue sequential recall of the words: "Recall of the general story theme cues the initial sentence, and recall of the sentence cues the target words within the sentence" (Hill, Allen, & McWhorter, 1991, p. 484).

While algorithms do not contain *words* per se, they do contain *procedural steps* that must be sequentially executed in order to compute a set of outputs from a set of inputs. Thus, given the results of studies of story mnemonics, it seems reasonable to hypothesize that constructing a personally meaningful story whose storyline maps to the procedural steps of an algorithm to be learned will lead to improved recall of the algorithm's procedural behavior. I shall return to the question of what "improved recall" might mean later on in the chapter.

## Graphics Creation Technique

Like most algorithm animation packages, the Samba package used in the ethnographic studies requires one to specify the position and appearance of objects within an animation in terms of (quantitative) Cartesian coordinates. For example, to position an object within an animation. one must specify two real-numbers—an x-coordinate and a y-coordinate. Recall that, in Chapter II, I labeled this kind of graphics creation technique *quantitative graphics*. In contrast, in constructing their storyboards, students typically created objects by either directly cutting them out of construction paper, directly sketching them on a page, or by using a direct-manipulation drawing editor. Likewise, students typically positioned objects in their storyboard by direct placement—using a hand, pen, or mouse. Let us call this alternative graphics creation technique *direct graphics*.

Observations made in Study II indicate that, in addition to *typeset fidelity and input generality, graphics creation technique* also greatly influences the relevance of student activities. When students created animations in Samba, they spent significant amounts of time figuring out the proper dimensions and coordinates of the objects in their animations. By contrast, when students used low-tech materials to create their homemade animations,

students no longer had to focus on graphics layout. Instead, they were able to concentrate more extensively on other issues, such as the procedural behavior of the algorithms they were animating.

## Self-Construction

A primary objective the ethnographic studies was to explore the benefits of AV construction assignments. Given this, it should not come as a surprise that *self-construction*—the construction of AVs by *students*, as opposed to experts—emerged as a significant factor in these studies. The more interesting question posed and partially answered by these studies relates to the nature of the benefits of self-construction: In what ways might self-construction be beneficial? In the studies, three key benefits of self-construction emerged.

The first key benefit has to do with the *process* it promotes. AV construction assignments engage students in an exploratory, creative process of constructing and refining personally-meaningful representations. According to cognitive constructivism (see Chapter III), this process of active learning is far superior to passively watching an AV, for it enables learners to actively construct for themselves the meaning and significance of the algorithms they are learning.

A second key benefit also stems from the process promoted by AV construction. The ethnographic studies suggest that the self-construction process gives students a personal stake in the algorithms they are learning; they become *vested* in the algorithms and the problems they are designed to solve. Articulating the sociocultural constructivist position, Lave (1997) labels this phenomenon *ownership*, and emphasizes its importance in the

learning process: "It is not possible [for students] to resolve problems that are not, in some sense, their own" (p. 33).

A third key benefit relates to the *products* of self-construction: animated representations of algorithms. The ethnographic studies indicate that student-constructed representations provide students with valuable resources for making their understanding of an algorithm accessible to others. And, as I explain below, student-instructor conversations involving such animated representations appear to play an integral role in helping students to learn algorithms.

## Instructor Communication

As just discussed, a key benefit of students' constructing their own AVs is that it helps students to make their conceptions and misconceptions of an algorithm accessible to others. The ethnographic studies demonstrate that, in subsequent discussions, student-constructed AVs appear to narrow the gap between expert and learner perspectives. An instructor can use a student's AV as a resource for understanding the student's perspective, and as a basis for clarifying concepts. A student can use her own AV as resource for explaining her perspective, asking questions, and interpreting an instructor's feedback and guidance.

Notice that, without subsequent interaction with others (most importantly with their instructor, but also with their peers), students miss out on an opportunity to obtain such valuable feedback and guidance on their emerging understanding. As a consequence, they have no way of knowing whether their view of the algorithm is on track; their misconceptions remain misconceptions, and their insights are not validated. Thus, as a follow-up to the self-

construction process, student-instructor interaction would appear to figure prominently in the effectiveness of AV-based pedagogical exercises.

## AV Effectiveness Measures

The previous section identified several factors that, in light of the results of the ethnographic studies, would appear to play a significant role in determining the effectiveness of pedagogical exercises involving AV artifacts. Yet, a fundamental question remains: Effective in what sense? Indeed, since the focus of the previous section was *cause*, I refrained from describing the effects in detail, or elaborating on the way in which the effects might be measured. In this section, I take a closer look at four key *effects* (which might also be called *measures*, or *benefits*) of AV-based pedagogical exercises that are brought about by the factors discussed in the previous section. These four measures are summarized in the taxonomy of Figure 23, and discussed in detail in the remainder of this section.

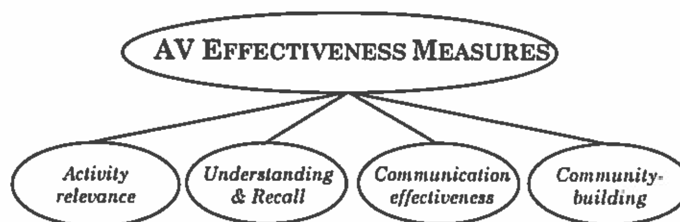



Figure 23. A Taxonomy of AV Effectiveness Measures

## Activity Relevance

Pedagogical exercises require students to engage in activities that can be more or less relevant to the particular course in which the pedagogical exercises are enlisted. For example, in the ethnographic studies, I found that AV construction assignments that require students to construct high epistemic fidelity AVs promote activities that are irrelevant to an undergraduate algorithms course, including graphics programming and graphics touch-up. On the other hand, the studies showed that, when asked to construct low epistemic fidelity storyboards, students engage in a higher concentration of relevant activities, including group design discussions and studying.

Thus, an important effect of pedagogical exercises is the relevance of the activities they promote, or their *activity relevance*. But what constitutes a relevant activity? Clearly, there exists no absolute measure of relevance. Rather, one must first conduct a field study in order to obtain as accurate a picture as possible of what students are up to, and then evaluate each student activity individually vis-à-vis the objectives of the course. Based on such an evaluation, one can then estimate an overall percentage of time students' spend on relevant activities versus irrelevant activities. The higher the percentage of relevant activities promoted by a pedagogical exercise, the better.

## Understanding and Recall

Thus far, the discourse on AV effectiveness has been preoccupied with a single gauge of effectiveness: *understanding*. This measure endeavors to assess how well a student learns an algorithm, upon engaging in a pedagogical exercise involving AV technology. As we saw in Chapter II, past AV effectiveness experiments have used written post-tests to

measure such understanding. The post-tests have traditionally included two kinds of questions, each designed to measure a distinct form of understanding:

1. *Procedural understanding:* Does the student understand *how* the algorithm works?
2. *Conceptual* understanding: Can the student make generalizations about the algorithm's behavior, including how efficiently it runs?

Past experiments have found that AV technology appears to have the greatest impact on procedural understanding (see, e.g., Byrne, Catrambone, & Stasko, 1996; Lawrence, 1993, ch. 9). This result is not surprising, given that AVs tend to illustrate how algorithms work, but not how efficiently or why.

Within the community of computer science educators, algorithm visualization tends to be regarded as a means to an end, with the desired end clearly being understanding: How well does the student *know* the algorithm? Recognizing the importance of this end, AV technologists have staged experiments that have focused squarely on measuring *algorithm understanding*. Indeed, for many instructors considering the adoption of AV-based pedagogical exercises, the desired proof-in-the-pudding is a well-designed experiment that demonstrates that AV-based pedagogical exercises promote better procedural understanding of an algorithm than alternative pedagogical exercises.

My fieldwork did not focus on assessing students' understanding of the algorithms for which they constructed and presented AVs. At the same time, the studies did not furnish evidence that speaks *against* the possibility that the AV construction assignments could have led students to develop an improved understanding of those algorithms. Because of the traditional importance of this measure within the computer science education community, and because the ethnographic studies do not rule the measure out, it seems an important one to include in this framework of cause and effect. The only looming questions

have to do with focus and operationalization: *What* exactly should we measure, and *how* should we measure it?

The results of past research, including my own observations, suggest that we would do well to refine the traditional operationalization of understanding in terms of procedural and conceptual knowledge. In constructing AVs that are supposed to illustrate *how* an algorithm works, students tend to focus intently on the *procedural* behavior of the algorithm: the sequence of steps it executes to transform inputs into outputs. Thus, at least in situations in which students do not discuss their AVs with an instructor, it seems safe to jettison *conceptual* understanding as a potential effect of both AV construction exercises, and of any pedagogical exercises that involve viewing an AV that illustrates *how* an algorithm works.

With the focus of the measure narrowed to procedural understanding, the question of how to measure procedural understanding arises. Given that AV-based pedagogical exercises require students to engage in activities in which they explore the step-by-step behavior of an algorithm, asking students to trace through that step-by-step behavior for a novel input data set would seem to be a suitable evaluation exercise. The accuracy of their traces with respect to an error-free trace, as well as the speed with which they perform the traces, could then serve as reasonable measures of procedural understanding.

Moreover, given that AV-based pedagogical exercises require students either to construct a graphical representation of the procedural steps of an algorithm (AV construction exercises), or to view a graphical representation of the steps for a variety of input data (AV viewing and interaction exercises), it seems reasonable to expect that such exercises would help students to *recall* the procedural steps of the algorithm. In the case of AV construction exercises, this effect might be predicted by past research into the value of stories as mnemonic devices, as discussed above. In the case of AV viewing exercises, on the other

hand, this effect might be predicted by past research into the value of *imagery* as a mnemonic device (see, e.g., Paivio, 1969). To measure recall, one could ask students to implement the algorithm in a programming language, and then compare their algorithm against an algorithm that is known to be correct.

## Effective Communication

As discussed earlier, a key benefit of AV construction assignments is that they require students to *represent* their understanding, and thereby to make it accessible to others. In subsequent presentation sessions, students' self-constructed representations enable students and instructors to bridge the gap between their perspectives—to develop a shared understanding of an algorithm. Recognizing the importance of representations in mediating such learner-expert interaction, Roschelle (1990) defines *symbolic mediation* as the use of a representation "as a resource for managing the uncertainty of meaning in conversations, particularly with respect to the construction of shared knowledge" (p. 1). On this view, the extent to which a representation is able to serve as a mediational resource determines, in large part, the effectiveness of the communication about the target concepts.

A second aspect of effective communication is that the concepts discussed in such conversations should actually be *target* concepts, as opposed to concepts that are peripheral to the course. For example, in the ethnographic studies, I found that students' Samba presentation sessions primarily generated conversations *about implementation details*—how a given feature of an animation was implemented, and what implementation difficulties were encountered, and the like. I also discovered that students' storyboard presentations generated conversations that were more sharply focused on issues relevant to an algorithms course, such as the aspects of an algorithm that are important to illustrate. Arguably, the

conversations about implementation details did not address the target concepts of the course, whereas the conversations about important aspects of an algorithm to illustrate did.

How does one measure these two aspects of effective communication: mutual intelligibility and topic relevance? *Conversation analytic techniques*, such as those used by Suchman (1987), scrutinize the structure and content of conversations in minute detail—utterance by utterance. The goal is to detect *communication breakdowns*—points in the communication at which a shared understanding is lost—as well as the subsequent repair of such breakdowns. Through such analysis, one can develop a fine-grained account of communication efficacy, firmly grounded in an empirical record (a transcript of the conversation). While such an account is qualitative, it nonetheless can provide insight into the extent to which representations serve as mediational resources in the establishment of shared understanding between conversational participants.

On the other hand, topic relevance can be assessed using the same general technique that I proposed to assess activity relevance. In particular, one can first obtain a fine-grained account, preferably a full transcription, of the conversations that take place during AV presentation sessions. Next, on an exchange-by-exchange basis, one can determine the relevance of the exchange with respect to the objectives of the course. Finally, one can compute topic relevance by determining the percentage of exchanges that were relevant, as opposed to irrelevant.

## Community-Building

At a macro level, what is going on within an undergraduate algorithms course? As discussed in Chapter III, sociocultural constructivism suggests that a distinct *community of practice* is actually in the process of reproducing itself. This reproduction takes place as

newcomers become old-timers by participating, in increasingly central ways, in the practices of the community.

Recall that, in Chapter III, I labeled the community that is in the process of reproducing itself through undergraduate algorithms courses the "Community of Schooled Algorithmaticians" (COSA). My fieldwork suggests that a well-defined participation structure was, in fact, in place in the classes I observed. Old-timers included the course instructor, and, to a lesser degree, the teaching assistant. Newcomers included the students enrolled in the course. Each of these actors participated according to established norms: The instructor participated through such activities as lecturing, holding office hours, issuing assignments, and giving exams, whereas students participated through such activities as attending lectures, studying the text, doing assignments, and taking exams.

If one accepts the sociocultural constructivist premise that, within an undergraduate algorithms course, a community of practice is in the process of reproducing itself, a new potential benefit of AV-based pedagogical exercises emerges. By providing occasions on which the community can come together to participate in meaningful activities, these exercises might contribute to the building up of the community, as well as to helping students to advance more rapidly from peripheral to central participation within the community.

But how might one measure such things as the existence of a community, and level of community membership? These are slippery notions that do not appear to lend themselves to measurement, at least not in a quantitative, operational sense. To be sure, it is not enough simply to measure individual performance in an activity that is relevant to the community, and to conclude that higher performance scores indicate a higher level of membership in the community. For one thing, such an approach assumes some sort of absolute that simply does not exist within a community, where determinations of knowledge

and good performance are largely a matter of agreement (see, e.g., Boster, 1985). Moreover, such an approach overlooks the importance of *identity* (the way in which one views oneself, and is viewed by others, with respect to the community) in determining level of community membership. Indeed, as sociocultural constructivism emphasizes, "learning and a sense of identity are inseparable: They are aspects of the same phenomenon" (Lave & Wenger, 1991. p. 115; see also pp. 52–53).

In sum, a potential benefit of AV-based pedagogical exercises is the extent to which the reproduction of the COSA is fostered, as well as the extent to which students come to participate, in increasingly expert ways, within the COSA. While this effect appears difficult, if not impossible, to measure (especially in a quantitative sense), it is nonetheless important, since an implicit goal of any algorithms course is to assist students in becoming competent members of the COSA.

## Hypotheses

The previous two sections outlined the central factors and measures in the AV effectiveness equation. One important question remains: What factors lead to what measures? Figure 24 puts the previous two sections together by causally linking factors to measures. If one groups these linkages by effect (the "Measures" column in Figure 24), four specific hypotheses emerge, each of which connects one or more causal factors to one of the four measures discussed in the previous section. In this section, I discuss these four hypotheses, and offer provisional explanations for them.

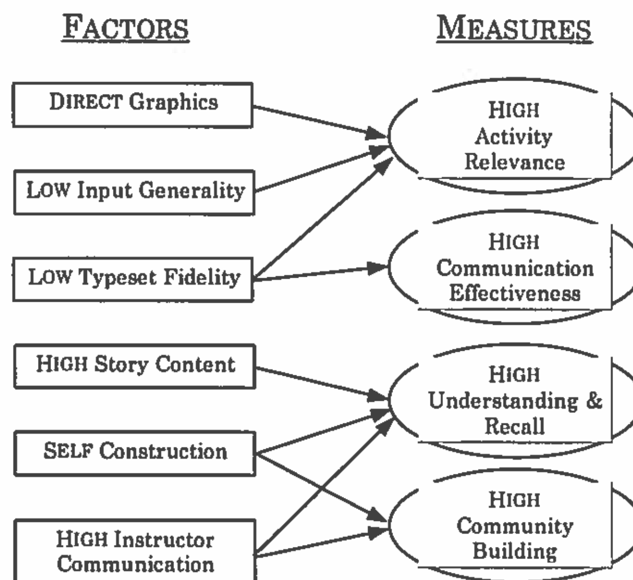



Figure 24. Graphical Summary of the Framework

## The Activity Relevance Hypothesis

The Activity Relevance Hypothesis speculates a causal link between epistemic fidelity and activity relevance, as illustrated in Figure 25.

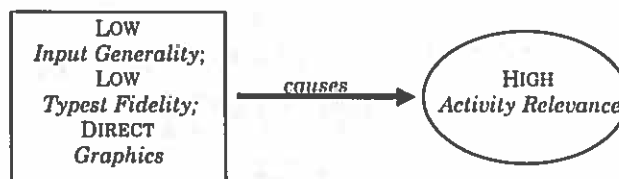



Figure 25. The Activity Relevance Hypothesis

As already discussed, when students construct *one-shot* (i.e., low input generality), low *typeset fidelity* AVs, they tend *not* to get bogged down in the (irrelevant) implementation details that they typically encounter when they construct high input generality, high typeset

fidelity AVs. As a result, not only do they spend significantly less time overall on AV construction, but they also spend a higher percentage of their time on activities that are relevant to the course—that is, activities that require them to focus them on the procedural behavior of the algorithms they are animating.

## The Communication Effectiveness Hypothesis

The Communication Effectiveness Hypothesis speculates the existence of a causal connection between typeset fidelity and communication effectiveness, as illustrated in Figure 26.

Figure 26. The Communication Effectiveness Hypothesis

In studies of architect-client interaction, rough, unpolished sketches have been shown to generate increased interaction regarding high-level design issues (see, e.g., Schumann, Strothotte, Raab, & Laser, 1996). Likewise, this hypothesis reasons that the rough, unpolished nature of low typeset fidelity AVs should encourage students and instructors to concentrate on the high-level issues behind the AVs, rather than on the aesthetics of the AVs, or on the details of how they were implemented. This shift in focus constitutes an increase in *topic relevance*, one of the two components of effective communication.

This hypothesis also reasons that the other component of effective communication, *shared understanding*, is enhanced by low epistemic fidelity AVs. In particular, as discussed

in Chapter IV, two main features of AVs appear to influence their utility as mediational resources:

1. *dynamic mark-up and modification*—an ability to be annotate, mark up, and modify an AV in response to clarifying questions; and
2. *execution control*—an ability to fast-forward, rewind, step through (at a flexible pace), and jump around within an AV.

Whereas low epistemic fidelity AVs can be easily annotated, marked up and modified, the high epistemic fidelity AVs created in systems like Samba cannot; modifying high epistemic fidelity AVs typically requires changing code, an activity that is not feasible within a presentation session. Furthermore, because they are under the complete control of their presenters, low epistemic fidelity AVs support a more flexible execution model; presenters can jump ahead in an AV, go back to a previous point in an AV, and step through an AV in response to the dynamics of the interaction they are having with their audience. In contrast, high epistemic fidelity AVs do not afford nearly as much flexibility; presenters are typically locked into a "tape recorder" model that supports only starting, stopping, and stepping (see Brown, 1988, p. 65).

## The Understanding and Recall Hypothesis

The Understanding and Recall Hypothesis posits that story content, self construction, and instructor communication are causally linked to recall and understanding, as illustrated in Figure 27.

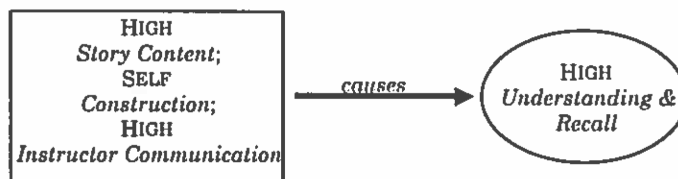



Figure 27. The Understanding and Recall Hypothesis

As discussed above, experiments have shown story construction to be an effective mnemonic device—one that helps people remember not only a list of words, but also the order of the words in the list. This empirical result suggests that the construction of a story that models the procedural steps of an algorithm should help one to recall the procedural behavior of the algorithm. Moreover, if one defines understanding as an ability quickly and accurately to trace through the procedural steps of an algorithm for novel input data, then one would expect understanding to go up as well, since improved recall of the algorithm's procedural steps should lead to increased speed and accuracy in tracing exercises.

A variety of explanations might account for the value of self-construction in promoting understanding and recall. As briefly discussed above, the sociocultural constructivist explanation emphasizes the role of AV construction assignments in giving students a sense of *ownership* in the algorithms for which they construct AVs (see Lave, 1997). Textbooks and lectures tend to own algorithms by presenting and analyzing them in vivid detail. For example, a typical textbook might provide a pseudocode description of an algorithm, perhaps augmented with a visualization, and then proceed to analyze the efficiency and prove the correctness of the algorithm. Since the textbook appears already to have figured out everything about the algorithm, it appears to *own* the algorithm. As a consequence, students may be pushed away, and lose incentive to explore the algorithm. By allowing students to create, develop, and instantiate their own representations of an algorithm, however, AV construction assignments shift ownership of the algorithm from the

textbook (and instructor) to the student. Such ownership *vests* students in the algorithm; they gain a personal stake in its meaning and significance. As a byproduct, their recall and understanding of the algorithm improves.

Also briefly discussed above, cognitive constructivism offers an alternative explanation that emphasizes the role of the AV construction process in facilitating students' active construction of their own understandings. In particular, by constructing and instantiating graphical models of an algorithm, students are required to actively articulate and put to the test their conceptualizations of the algorithm's procedural behavior. In the process, they must reorganize and adapt their understanding of the algorithm's procedural behavior so as to resolve any inconsistencies or holes that they notice in the representations they are building.

While constructing an AV may improve a student's understanding and recall of the underlying algorithm, it does not guarantee that the understanding so developed approaches that of an expert. Through a subsequent presentation session with an instructor, however, a student has the opportunity to gain crucial feedback and guidance. Such feedback and guidance can help the student to refine her understanding of the algorithm, leading to a view of the algorithm that more closely resembles that of the instructor.

Note that the causal link posited between student-instructor interaction and understanding and recall applies to *any* AV, regardless of whether it is constructed by a student or constructed by an expert. In the case of an AV constructed by an expert, student-instructor interaction plays an essential role: it enables students and instructors to *negotiate* the meaning and significance of the AV, neither of which is self-evident (see, e.g., Suchman, 1987). In the case of an AV constructed by a student, on the other hand, this hypothesis predicts a positive interaction effect involving self-construction and interaction. In particular, students who construct their own AVs place themselves in a position in which

they can maximally benefit from subsequent interaction they have with an instructor regarding their AVs. Thus, the interaction they have with an instructor contributes more to improved understanding and recall than it otherwise would.

## The Community-Building Hypothesis

The Community-Building Hypothesis suggests that the combination of self-construction and interaction within AV-based pedagogical exercises leads to the building up of a community, as illustrated in Figure 28.

Figure 28. The Community-Building Hypothesis

This hypothesis accounts for the effectiveness of AV technology in terms of the sociocultural constructivist view of learning introduced in Chapter III. As discussed there, AV artifacts can be seen as facilitating three key forms of participation within the community that is in the process of reproducing itself through undergraduate algorithms courses (*viz.*, the *COSA*):

1. interpreting and making sense of graphical representations of concepts and themes surrounding algorithms (*AV reading*);
2. constructing such representations (*AV construction*); and
3. using such representations as the basis for presentations and discussions (*AV-mediated communication*).

Notice that AV construction assignments provide opportunities for students to participate in all three of those activities, whereas traditional AV-based pedagogical exercises provide opportunities to participate in only one of them (AV reading). Therein, according to sociocultural constructivism, lies the primary benefit of self-construction and interaction: they provide access to increasingly central (expert) forms of participation—forms of participation to which students have not traditionally had access.

AV presentation sessions like the ones described in Chapter IV yield a secondary benefit as well: opportunities for observation and feedback. The interaction between students and instructors that takes place provides important clues to appropriate ways of presenting, discussing, and commenting on visual representations of algorithms. Thus, in addition to participation, observation would also seem to facilitate students' centripetal movement in the community—from the outer, peripheral layers of the "community onion," to the inner, core layers of the "community onion."

It is important to emphasize that this process of advancement through the participation structure defined by a community not only enables the community to reproduce itself, but also, on the sociocultural constructivist view, *constitutes learning in itself*. Observe that this community-based notion of learning differs markedly from the conventional, cognitive-based notion of learning exemplified by the Understanding and Recall measure (see above). Indeed, coming to participate, in increasingly central ways, within a community of practice encompasses far more than acquiring procedural understanding and recall; it also entails the development of skills and expertise that are beyond those that can be evaluated by conventional written tests, including (a) an ability to effectively explain an algorithm to someone else (which, in turn, involves recognizing what aspects of the algorithm are important to illustrate, and what grain of analysis is appropriate); (b) an ability to critique explanations of algorithmic behavior at appropriate times; (c) an ability to ask appropriate

questions about algorithms at appropriate times; (d) interest and motivation to explore algorithms; and (e) confidence in and comfort with the discourse of algorithms.

In sum, sociocultural constructivism provides a theoretical foundation for the hypothesis that self-construction and interaction lead to the maintenance of a community of practice in which students advance toward fuller membership. On this view, AV construction exercises succeed precisely because they provide access both to key forms of COSA participation (AV construction and presentation), and to old-timers and newcomers engaged in presenting and discussing algorithms.

## Summary and Research Directions

This chapter has taken the ethnographic studies presented in the previous chapter as a point of departure for pinpointing key causes and effects in the AV effectiveness equation, and for positing a series of hypotheses that link them together. The goal has been to translate the findings of the ethnographic studies into a specific theoretical position that future research can put to the test.

Resonant with both cognitive and sociocultural constructivism, the theoretical position that emerges out of the framework stresses the value of students' constructing and refining their own personal representations of the material they are learning. At the same time, in line with sociocultural constructivist theory, the position emphasizes the importance of students' participating more centrally in algorithms practice by presenting their self-constructed AVs to their peers and instructor for feedback and discussion. Finally, in radical departure from AV technology's obsession with high epistemic fidelity visualizations, the position underscores the value of low epistemic fidelity visualizations in focusing students' on algorithms and how they work, and in promoting effective communication about algorithms.

This theoretical position can thus be seen as a specialization of sociocultural constructivist learning theory, rather than as a broadening of EF Theory. Indeed, whereas constructivism well accounts for all four framework hypotheses, EF Theory stands a chance of accounting for only one of them: the Understanding and Recall Hypotheses; the other three hypotheses lie beyond its analytical scope. Thus, if the ethnographic findings that inspired the framework can be generalized beyond the particular algorithms course that I studied , sociocultural constructivism should prove to be a more suitable theoretical foundation for guiding future AV research.

In addition to articulating the beginnings of an alternative theoretical position, the framework presented in this chapter has aimed to identify issues that are grist for further investigation. The framework indeed raises numerous research questions, which I enumerate in Chapter VIII. Within the scope the scope of this dissertation, however, I limit my focus to two of the most important research issues, which I explore in the next two chapters:

1. *Empirical validation of the hypotheses.* All four of the framework's hypotheses were motivated by qualitative evidence collected in the ethnographic studies presented in the previous chapter. An obvious question to ask is, Can any of these hypotheses be validated through a more rigorous controlled experiment? Because of its similarity to the hypotheses tested in past experiments of AV technology effectiveness, the Understanding and Recall Hypothesis is a prime candidate. In Chapter VI, I propose a series of experiments for validating the Understanding and Recall Hypothesis, and I present an actual experiment that put one component of the hypothesis to the test.
2. Design implications of the hypotheses. The framework's hypotheses have profound implications for the design of AV technology. If one takes these

conjectures as constraints on the design space of effective AV technology, what kind of AV artifact emerges? This question, which should be of great interest to AV technologists, is pursued in earnest in Chapter VII, in which I describe the design of a prototype AV artifact rooted in the hypotheses.

# CHAPTER VI

# EXPERIMENTAL STUDIES

Science is a willingness to accept facts even when they are opposed to wishes.

B.F. Skinner [as quoted in (Martin, 1996, p. 88)]

The legacy of controlled experiments reviewed in Chapter II aimed to validate the pedagogical effectiveness of AV technology. They did so by defining effectiveness quite narrowly in terms of procedural and declarative understanding of algorithms. As sketched out by the framework of cause and effect presented in the previous chapter, sociocultural constructivist theory, backed by the ethnographic findings presented in Chapter IV, suggests a far broader definition of effectiveness. Even though this definition is broader, notice that it still includes a rather traditional notion of "understanding and recall," one that greatly resembles the notion of understanding proposed by past experiments. Given that past empirical research has pursued evidence in support of AV technology's ability to foster understanding, and given the value that computer science instructors place on traditional learning outcomes[30], it makes sense to focus first on those framework factors that are posited to influence understanding and recall. Moreover, given that controlled experimentation has

---

[30]This value became remarkably clear to me when I presented, at a well-known professional conference that included computer science educators, an earlier agenda for this research. At that time, I was interested in pursuing research that explored learning outcomes at a community level, in line with the Community Building Hypothesis presented in Chapter V. However, my audience made it clear to me that what computer science instructors really care about was not the extent to which students gain membership in a community, but rather the extent to which students "learn" an algorithm. After that conference, I revised my research agenda to include the experiments discussed in this chapter, which examine more traditional learning outcomes.

become the de facto standard for evaluating the pedagogical benefits of AV technology, it makes sense to employ controlled experiments as a means of exploring those factors.

In this chapter, I propose a series of experiments that systematically explore three factors that, according to the framework of the previous chapter, influence understanding and recall. I then present the results of the first experiment of this series, which I conducted within the scope of a later offering of the same CIS 315 course in which I conducted the ethnographic fieldwork presented in Chapter IV. In exploring the influence of self-construction on understanding and recall, this experiment essentially tests the cognitive constructivist hypothesis that learners must construct their own understandings, rather than "absorbing" the knowledge of someone else. The remainder of the experiments in the series, which are left for future work, would explore two other factors that are posited to influence understanding and recall: instructor communication and story content.

## A Series of Experiments

Figure 29 presents the "Understanding and Recall" hypothesis proposed in Chapter V. This hypothesis causally links three factors (independent variables) to two measures (dependent variables), as elaborated in Table 5.

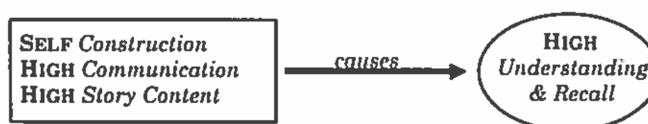



Figure 29. The Understanding and Recall Hypothesis

Table 5. Independent and Dependent Variables in the Understanding and Recall Hypothesis

| Independent Variables | Dependent Variables |
|---|---|
| 1. self-construction—whether or not students construct their own AVs.<br>2. instructor communication—whether or not students have the opportunity to discuss the AVs with an instructor.<br>3. story content—whether or not the AV portrays the algorithm in terms of a story. | 1. procedural understanding—how quickly and accurately a student can perform a detailed trace of an algorithm's procedural behavior.<br>2. recall—how quickly and accurately a student can write a procedural description of the algorithm in a programming language |

To explore the influence of self construction, instructor communication, and story content on procedural understanding and recall, I propose a series of three controlled experiments (see Table 6). Each experiment in the series manipulates one of the factors while holding all other framework factors[31] as constant as possible. Moreover, as the arrows in the figure indicate, the results of previous experiments serve as input to later experiments by determining the best values of the independent variables.

Table 6. Series of Planned Experiments

| | | *Independent Variables* | | | | |
|---|---|---|---|---|---|---|
| Experiment # | Treatments | Self-Construction | Instructor Communication | Story Content | Input Generality | Typeset Fidelity |
| 1 | Self-Construction | Yes | No | Yes | Low | Low |
| | Active-Viewing | No | No | Yes | Low | Low |
| 2 | Instructor | ??? | Yes | Yes | Low | Low |
| | No Instructor | ??? | No | Yes | Low | Low |
| 3 | Story | ??? | ??? | Yes | Low | Low |
| | No Story | ??? | ??? | No | Low | Low |

[31]Note that I do not propose to manipulate input generality and typeset fidelity because I do not believe them to have a significant influence on understanding and recall.

In the following section, I describe and present the results of the first of these experiments.

## Experiment 1: Active Viewing versus Self-Construction

The ethnographic fieldwork presented in Chapter IV explored a constructivist approach to using AV technology as a pedagogical aid. First explored by Stasko (1997), that approach has students construct their own AVs using conventional AV software. As I argued in Chapter IV, the results of the ethnographic studies suggest an improvement to that approach: namely, that students should construct their own AVs with low epistemic fidelity materials (e.g., pens, paper, scissors), rather than with conventional (high epistemic fidelity) AV software.

The objective of the first experiment in the series was to evaluate empirically this improved constructivist approach by pitting it against the most pedagogically effective EF-minded approach, as determined by Lawrence's (1993) experiments. As discussed in Chapter II, of all the factors that Lawrence considered, active viewer involvement had the most significant effect on learning outcomes. Specifically, students who interacted with predefined AVs by designing their own input data sets and then observing the AVs operate on those input data sets significantly outperformed students who either interacted with predefined AVs operating on predefined input data sets (Lawrence, 1993, ch. 6), or who did not interact with AVs at all (Lawrence, 1993, ch. 9).

Thus, the specific hypothesis tested in Experiment 1 was that students who construct their own AVs will obtain a better procedural understanding, and better recall abilities, than students who view a predefined AV operating on their own input data sets:

Hypothesis: On tracing and programming tasks designed to assess procedural understanding and recall, students who construct their own AVs will outperform students who actively view a predefined AV.

## Design

To test this hypothesis, Experiment 1 employed a between-subjects design with two treatments: Self Construction and Active Viewing (see Table 7).

Table 7. Design of Experiment 1

| Treatment | # Participants |
|---|---|
| Self Construction | 11[32] |
| Active Viewing | 12 |

In the Self Construction treatment, students learned about an algorithm by constructing their own (low epistemic fidelity) AVs out of simple art supplies. In the Active Viewing treatment, students learned about the same algorithm by designing their own input data sets and viewing a predefined AV operate on those input data sets. Participants' learning outcomes were assessed according to four dependent measures: (a) trace accuracy, (b) time to construct trace, (c) program accuracy, and (d) time to construct program. I will detail the way in which I evaluated participant performance on these measures later on in the chapter.

Participants were assigned to treatment groups such that the two treatment groups were optimally matched according to self-reported grade point average.[33] To counterbalance

---

[32]Originally, 12 participants were in this treatment; however, one of the participants withdrew from the study before completing all of the tasks. I thus elected to drop this participant's data.

any potential task order effects, I further divided participants in each treatment into two groups, such that half the participants in a given treatment completed the tracing task before the programming task, and the other half completed the programming task before the tracing task.

## Equalizing the Treatments

To ensure that any observed effects were due to the manipulation in the independent variable (construction), I had to take great care to hold the other four independent variables hypothesized to be significant (*viz.*, instructor communication, story content, input generality, typeset fidelity; see Table 6) as equal as possible in the two treatments. In the case of instructor communication, that was easy; students in both conditions were not given the opportunity to communicate with an instructor about the AVs they were exploring. In the case of the other three independent variables (which I labeled "representation" variables in Chapter V), however, equalizing the treatments proved trickier. Before I turn to a detailed description of the experiment, it is worth discussing the measures I took to equalize story content, input generality, and typeset fidelity in the two treatments.

As Table 6 indicates, I chose to have story content in both treatments. In other words, both treatments were to work with AVs that portrayed the algorithm in terms of a story. Since I created the AV used in the Active Viewing treatment, I could make sure that the AV portrayed the algorithm in terms of a story. In the Self Construction treatment, by contrast, students were required to create their own AVs, so I ultimately had little control over whether their AVs had story content. Indeed, prior empirical research suggested that it

---

[33]The average difference, in grade points (on a 4 point scale), between matched pairs was 0.08. This does not include the pair containing the participant who dropped out.

was not a matter of simply "requiring" students to construct an AV that portrayed the algorithm in terms of a story. For one, my fieldwork indicated that devising a story is challenging; few students chose to do it. Furthermore, in the pilot study I ran prior to the experiment, I found that if the experimental instructions emphasize that student AVs must tell a story, students might become so sidetracked trying to come up with a story that they have little time to study the algorithm. In light of these observations, I worded the instructions carefully as follows (see "Instructions: Algorithm Animation Construction Exercise" in Appendix D):

> Coming up with a story to illustrate any algorithm is challenging! Please don't get hung up for too long trying to come up with a personally-meaningful story that illustrates the algorithm. If, after a few minutes, you still haven't come up with a story. then just go with whatever visual representation of the algorithm comes to mind. If you can come up with a story, then great! But for this exercise, actually constructing a homemade animation, even if it is not based on a story, is more important.

The input generality factor presented the second equalization challenge: How could one set input generality to "low" in both conditions, given that the pre-defined AV had to be capable of illustrating the algorithm for general input, and given that a self-constructed AV could only illustrate the algorithm for those input data sets that its creator chose? My solution to this dilemma was to word the experimental instructions for both treatments equivalently. For the Active Viewing treatment, the instructions stated that students should view the animation for at least five input data sets. For the Self Construction treatment, on the other hand, the instructions stated that students should walk through their homemade animation for at least five input data sets. In both cases, students were free to explore more input data sets as time allowed. Thus, students had the opportunity to study an equivalent range of input data sets in both treatments, although it is important to note that students in the Active Viewing treatment generally explored more input data sets because of the lower overhead required to do so.

The final representation variable that I had to take care to equalize was typeset fidelity—that is, extent to which the AVs in the two treatments resembled the polished look of textbook figures. In the Self Construction treatment, students worked with simple art supplies to create "scruffy" AVs with a low level of typeset fidelity. In the Active Viewing treatment, I created an AV that had the lowest possible typeset fidelity, given that it was created with conventional AV software and that it had to work for general input. Specifically, I made sure that the elements of the AV were simple and unpolished stick figures. In fact, a post-hoc analysis of the low-fidelity storyboards developed by participants in the Self Construction treatment suggests that the AV explored by the Active Viewing treatment had a level of typeset fidelity that was equivalent to that of the AVs produced by the Self Construction participants.

## Participants

I recruited 23 students (6 female, 17 male)[34] out of the winter, 1999 offering of CIS 315, "Algorithms," at the University of Oregon. This offering of CIS 315 had nearly the same content and structure, and was taught by the same instructor (Professor Lane), as the two courses in which I conducted the ethnographic studies reported in Chapter IV. Student participants were all majoring in computer and information science, and had an average self-reported average cumulative grade point average of 3.12 (on a 4 point scale).[35] I paid them a

---

[34]Actually, I recruited 24 students (7 female, 17 male), but recall from footnote 32 that one student dropped out of the study before completing it, so I only collected data for 23 participants.

[35]I requested that they verify their cumulative grade point average before reporting it to me by logging on to the university registrar's web-based student information system. Thus, while I could not look over their shoulders as they did this, I have good reason to believe that the numbers they reported were accurate.

$50 honorarium for their participation in the study. See Appendix D for a copy of the "Informed Consent Agreement" that they signed.

## Materials

Participants learned about the QuickSelect algorithm (see, e.g., Cormen, Leiserson, & Rivest, 1990, pp. 153–155, 185, 187–188), which had been covered in a CIS 315 lecture prior to the study.[36] The QuickSelect algorithm uses the same divide-and-conquer strategy used by the well-known QuickSort algorithm. However, instead of sorting a list of elements, QuickSelect finds the $i^{th}$ smallest element in a list. The CIS 315 course textbook (Cormen, Leiserson, & Rivest, 1990) calls this the $i^{th}$ *order statistic.*

During the learning phase of the study, participants in both treatments were given a set of instructions (different for each treatment group) describing how to use AV technology to learn the algorithm, as well as a three-page description of the QuickSelect algorithm, including the Partition procedure on which it intimately relies.[37] It is important to note that the Partition procedure explained in the three page description differed from the Partition procedure covered in the CIS 315 textbook. The biggest difference was that the Partition procedure explored in the study always selects as the pivot element the *first* element in the array to be partitioned, whereas the CIS 315 textbook's version of Partition chooses a random pivot element. I purposely chose a different version of the algorithm in order to

---

[36]Since it took about a week and a half to run all participants through the study, some participants had heard the QuickSelect lecture more recently than others; however, all participated in the study within three weeks of attending the lecture.

[37]Note that copies of all experiment materials can be found in Appendix x.

ensure that students who thought they already "knew" the algorithm would have to revisit it within the study.

Active Viewing Treatment Group

Participants in the Active Viewing group interacted with an input-general AV developed on top of Microsoft Windows with the Polka animation package (Stasko & Kraemer, 1993). The animation explains the QuickSelect algorithm through the Story of the Forest Rangers:

> Head Ranger Rhonda has just been notified by her bosses at the National Forest Service that a stand of trees is to be clear-cut. However, Rhonda is also told that $i^{th}$ shortest tree in the stand can be spared; it is her job to find that tree. With the help of her trusty assistants, Junior Rangers Rodham and Rachel, Rhonda proceeds to apply the QuickSelect algorithm to the stand of trees in order to find the tree to be spared from the clear-cut.

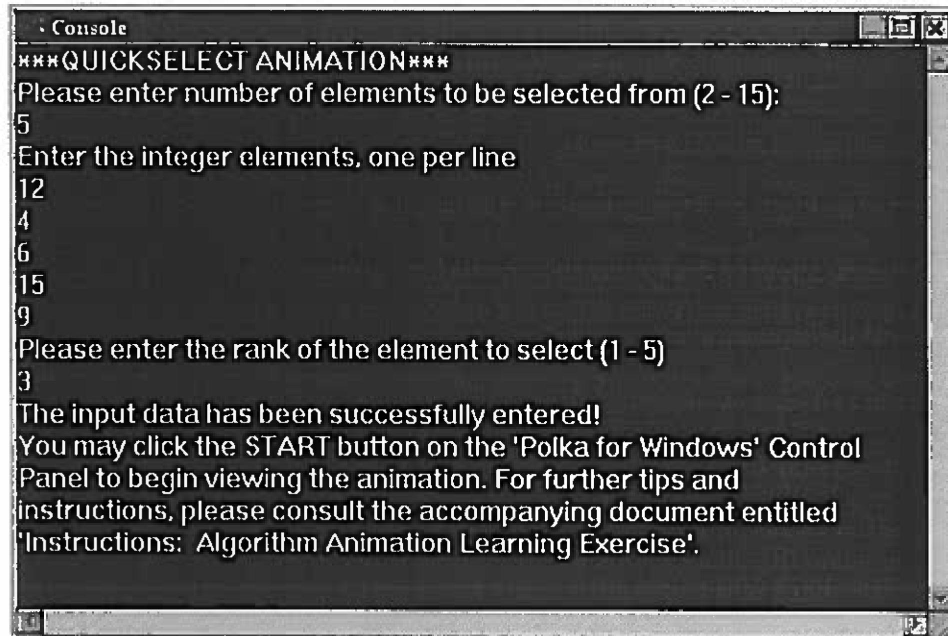



Figure 30. The "Console" Window

Upon launching the animation, one uses the Console window (see Figure 30) to enter (a) the number of elements (trees) to select from, (b) the individual values (heights) of the elements; and (c) the rank of the tree to select—that is, the rank of the tree to spare from the clear-cut. One can then use the Polka Control panel (see Figure 31, which includes annotations describing what each button does ) to control the execution of the animation as it operates on the input data that has been entered.

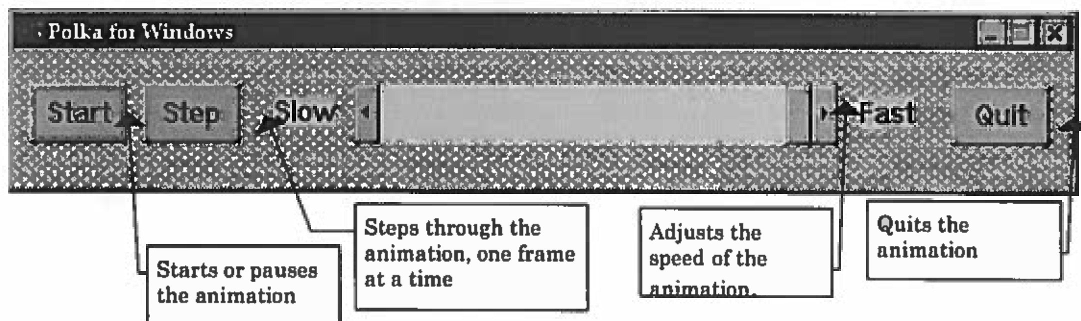



Figure 31. The Polka Control Panel

The main action of the animation unfolds in the "Forest Rangers" window (see Figure 32 for a snapshot; the annotations describe what the buttons at the bottom of the window do). In that window, Rhonda introduces the plot of the story, and, with the help of her assistants, Rodham and Rachel, walks through the execution of the QuickSelect algorithm for the input data that was previously specified. Each ranger has a speaking bubble, which flashes when he or she talks. In addition, an explanatory caption across the top of the window summarizes each key step of the algorithm as it unfolds. In response to feedback I received in the pilot study, I added this caption to help viewers coordinate the action of the animation with the pseudocode.

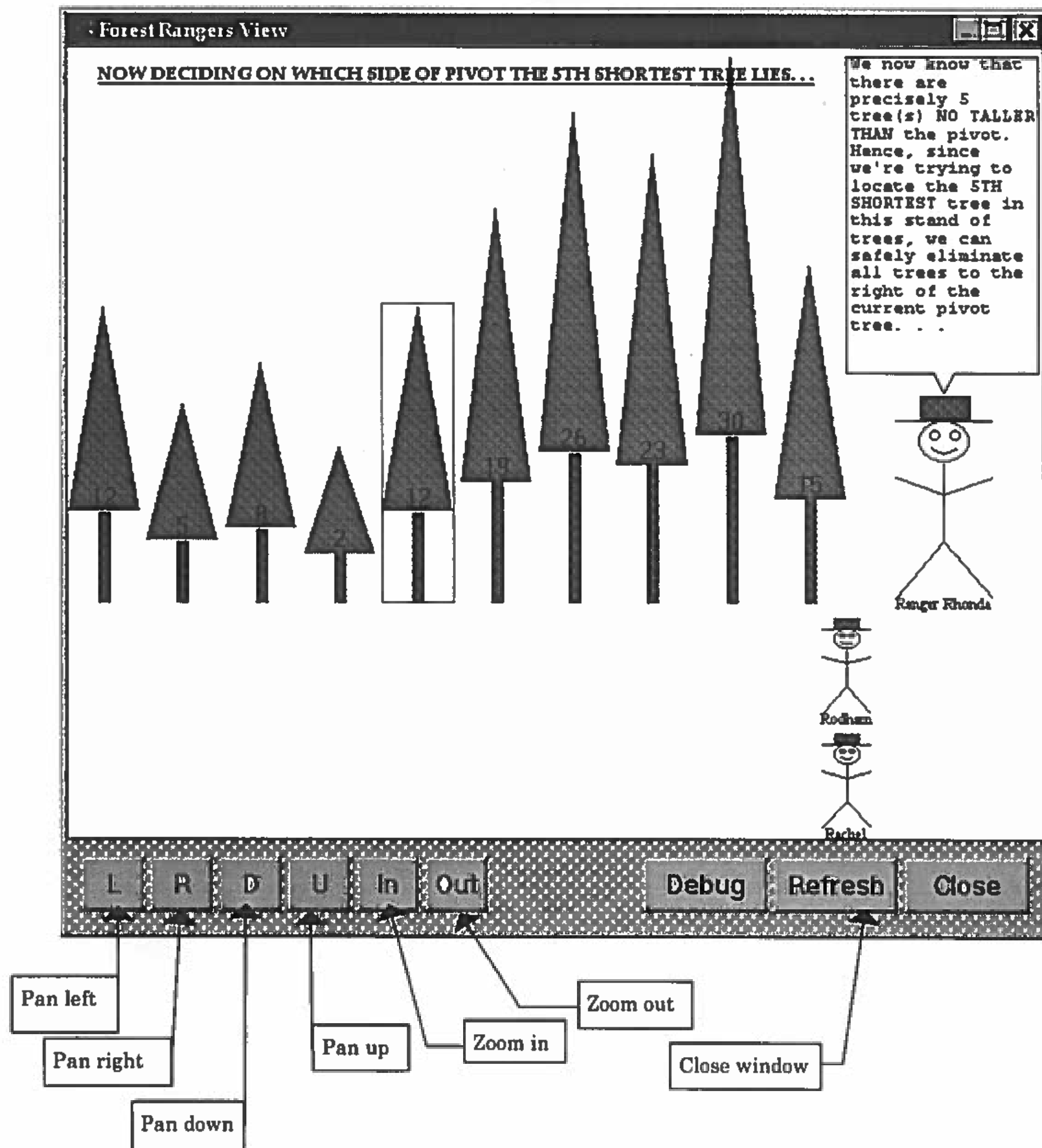



Figure 32. The "Forest Rangers" Window

In an informal programming walkthrough (Bell, Rieman, & Lewis, 1991) of the tracing tasks that participants were required to perform, I discovered that, while participants could glean from the Forest Rangers window the information they would need to perform the tracing tasks, a supplementary view could prove helpful by making the

necessary information more explicit. Consequently, the animation also included an ancillary "History" window (see Figure 33 for a snapshot), which presents a summary of all of the key algorithm steps that have occurred up to the current point in the execution of the Forest Rangers window. As clarified by both the annotations in Figure 33, and by the "History Legend" window (see Figure 34) included as part of the animation, each row in the *History* window summarizes the key data values of, and decisions made within, one recursive call to the QuickSelect algorithm. The arrangement of the trees within the row indicates the state of the array *after* the call to Partition within that recursive call. Essentially, the History view furnishes the same kind of trace that participants were required to perform within the tracing tasks.[38]

Self Construction Treatment Group

Participants in the Self Construction group were given a packet containing the raw materials for the homemade animatons they were to construct: a variety of art supplies, including colored pens, scissors, glue sticks, scotch tape, construction paper in a spectrum of colors, colored notecards, and white scratch paper.

---

[38]In fact, I used the History window to construct the answer keys with which I scored participants' traces.

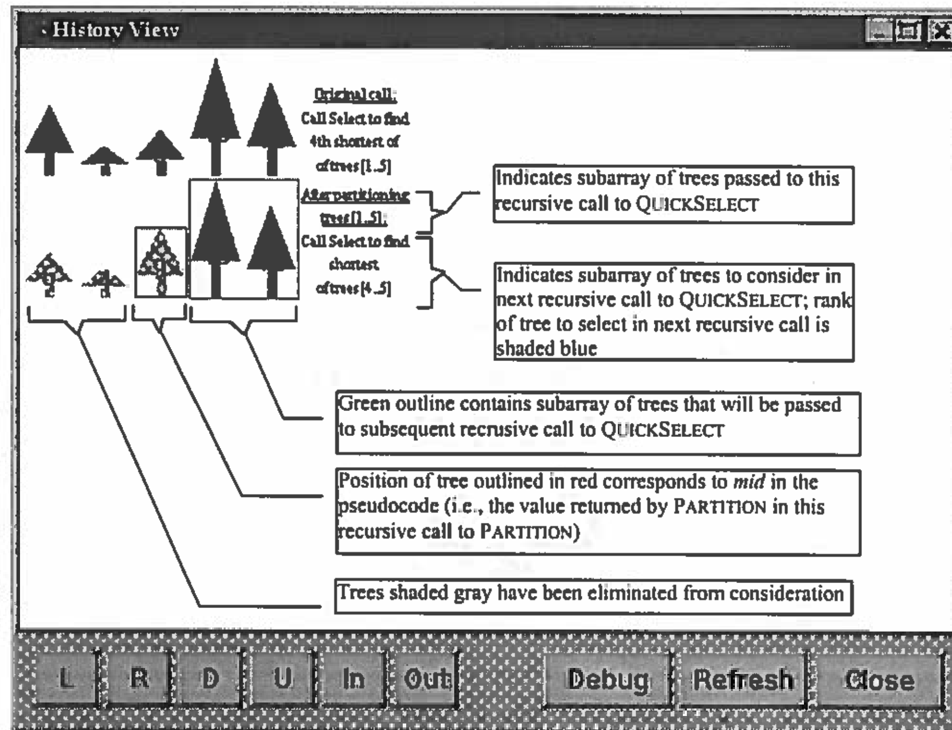



Figure 33. The “History” Window

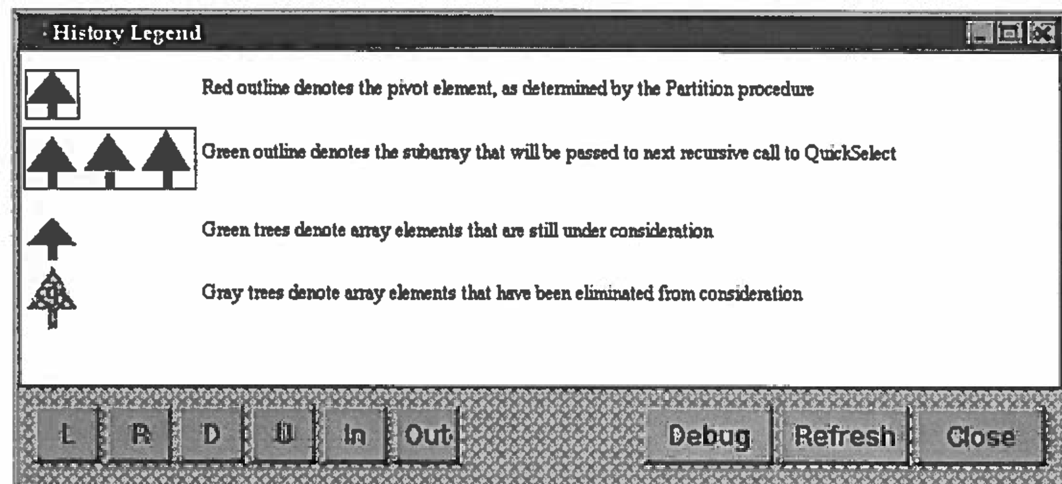



Figure 34. The “History Legend” Window

## Tasks

In the learning phase of the study, each treatment group was given the same two explicit learning objectives (see Appendix D):

Upon completing this part of the study, you should be able to

1. perform a detailed trace of the QuickSelect algorithm (including the Partition procedure) operating on any input data set; and

2. remember the QuickSelect algorithm (including the Partition procedure) well enough to implement it in a programming language, if you had to.

In addition, each treatment group received detailed instructions on how to proceed. Along with a complete description of the Forest Rangers animation, the Active Viewing group was given step-by-step instructions on how to use the animation. These instructions recommended that they run the animation on at least five, carefully chosen input data sets, and that they study the "History" window each time the animation runs to completion.

The Self Construction group, on the other hand, received instructions on how to create a homemade animation out of art supplies. To get participants started with their homemade animations, the instructions posed and elaborated on three guiding questions:

1. What is important about the QuickSelect algorithm?

2. What to draw? What to say?

3. What story to tell?

Using these questions as a framework, I made the instructions more concrete by presenting an example of the kind of homemade animation participants were to create. My example demonstrated how one might develop a story-based animation to illustrate the well-known bubblesort algorithm.[39] I summarized the process thus:

---

[39]The Football animation I presented was adapted from one observed in an earlier empirical study (Douglas, Hundhausen, & McKeown, 1995).

> Think of the homemade animation you'll be developing as an exercise in "implementing" a general-purpose graphical simulation. Your animation is general-purpose because you should be able to walk through the animation for any input data set you choose.

Self Construction participants were instructed to walk through their homemade animation for at least five input data sets. So as to conform to the limitations of the Forest Rangers animation, these input data sets were to have a maximum of 15 data elements, with data element sizes ranging from 1 to 100. As they walked through their animation, they were asked to think or say to themselves the play-by-play narrative they would use to explain how the algorithm works. Moreover, in order document that they had actually instantiated their animations on five input data sets, and in order to place them on equal footing with Active Viewing participants, Self Construction participants were asked to use an 8½" by 11" sheet of paper to create a "history view" for each of their five input data sets. Just like the "History" window included in the Forest Rangers animation, their history views were to summarize each execution of their animation by including "snapshots" at key points in that execution. In my demonstration of how to construct a homemade animation, I presented an example of what a history view might look like.

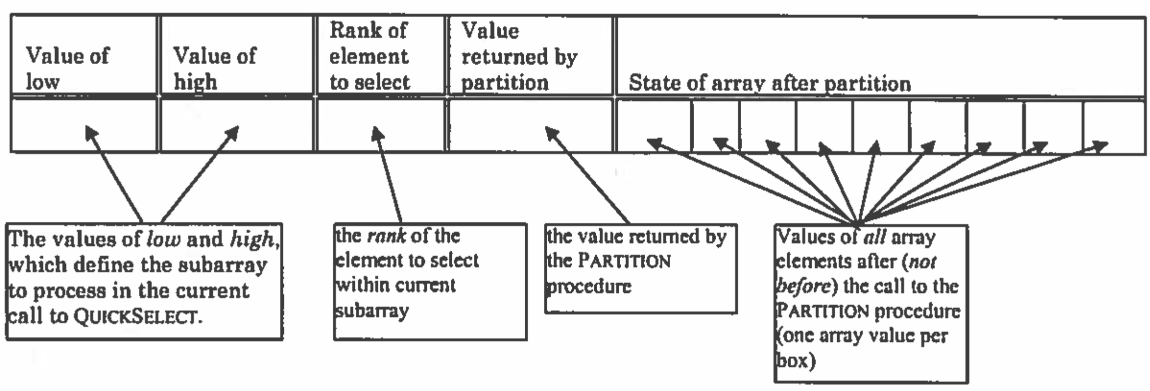
In the evaluation phase of the study, all participants performed the same two tasks. For the tracing task, participants were asked to trace through the execution of the QuickSelect algorithm for four different input data sets. I selected the input data sets so as to cover a broad range of possible inputs: average case, best case, worst case, and degenerate case.[40] Participants documented their traces by using Microsoft Word to fill in a table like the one presented in Table 8 (see the annotations for explanations of each entry). Each row they filled in was to correspond with one recursive call to the QuickSelect algorithm. Notice

---

[40]Because of idiosyncrasies in the Partition procedure explored in the study, there existed certain inputs for which the QuickSelect algorithm got stuck in an infinite loop. All of these degenerate case inputs contained duplicate elements. (However, not all input data sets containing duplicate elements caused the algorithm to get stuck.)

that table entries correspond quite closely with the summary information provided by the "History" window included in the AV used by the Active Viewing treatment.

Table 8. The Table Participants Filled in for Each Trace




For the programming task, participants were given a partially complete specification of the QuickSelect algorithm written in either C++ or Java (they could choose which one). They were instructed to fill in the missing code, which included the following three segments:

1. The recursive *select* method;
2. The *partition* method; and
3. The parameter list in the original call to the `select()` method.

The skeleton programs that they edited were extensively documented, and places where code was to be filled in were clearly marked with ***TO DO*** blocks; see Appendix D for printouts of the programs.[41]

---

[41]While it does not show up in these printouts, the programs that participants edited were color-coded to enhance their readability. Keywords were colored blue, comments were colored green, and the ***TO DO*** blocks were shaded red.

## Procedure

Experimental sessions were divided into two distinct phases. In the learning phase, participants had two and a half hours to study the QuickSelect algorithm by using the description of the QuickSelect algorithm, along with AV technology (either the Forest Rangers animation or art supplies).[42] Upon completing the learning phase of the study, participants in both treatments entered the evaluation phase of the study, in which they completed the tracing and programming tasks. Participants used Microsoft Word for Windows to complete both of these tasks on PC computers. They had a maximum of 25 minutes to complete each of the four tracing tasks, and a maximum of 35 minutes to complete the programming task.[43] After completing the tracing and programming tasks, participants were asked to reflect on their experiences by completing a brief exit questionnaire (see Appendix D).

## Scoring the Traces and Programs

I developed simple systems for evaluating trace accuracy, program accuracy, trace time, and program time. To evaluate tracing accuracy, I created answer keys containing perfect traces for the four tracing tasks. I then assigned one point to each correct entry in the "perfect" table for low, high, rank of element to select, and value returned by Partition (see Table 8).[44] With respect to the state of the array after Partition, I assigned one point to

---

[42]Participants were given a 30 minute break two hours into the learning phase.

[43]These time limits were determined in the pilot study I conducted prior to the experiment.

[44]Note that participants got "for free" the initial values of low, high, and rank of element to select in the first recursive call, so those values did not count.

each changed value in the array, and an additional point total to the remaining unchanged values. Thus, the number of correct points for a given trace depended on the total number of rows (one for each recursive call), and the number of changed array values in each of the rows. Trace 1 (average case) was worth 40 points, Trace 2 (best case) was worth 11 points, Trace 3 (degenerate case) was worth 32 points, and Trace 4 (worst case) was worth 63 points.

To evaluate programming accuracy, I used the CIS 315 textbook's (Cormen, Leiserson, & Rivest, 1990) pseudocode description of the Partition procedure and the recursive Select procedure to develop a list of semantic elements for each procedure. Essentially, a semantic element constituted a line of pseudocode, including the parameter lists to the two procedures. I assigned a varying number of points (1 – 3) to each semantic element, depending on its composition and complexity, and assigned participants' programs a score based on the number of correct semantic elements they contained. While I did deduct points for semantic elements that were not ordered correctly, I did not mark off for minor syntax errors. The recursive Select procedure and the Partition procedure were weighted equally, each being worth 15 points. I gave an additional point for the correct initial call to the recursive Select procedure, bringing the total number of possible points for the programming task to 31.

Finally, since participants completed both the tracing and programming tasks in Microsoft Word, I used the "created" and "modified" file timestamps automatically generated by Microsoft Windows operating system to compute accurate task times.

# Results

Table 9 presents the mean trace scores and times of the two treatment groups for each of the four traces, along with the combined average percentage-correct for all four traces. Figure 35 and Figure 36 present box plots of the scores and times. In these figures, the boxed regions delineate the range of scores and times for each treatment group; the horizontal line through each box marks the median. While the Self Construction participants scored slightly higher, a two sample *t*-test found no significant difference between the two treatments ($t = -0.211$, $df = 21$, $p < 0.83$). Likewise, while the Self Construction participants took slightly longer, on average, to complete the tracing tasks, a two-sample *t*-test determined that the difference was not significant ($t$ = -0. 0.353, $df = 21$, $p < 0.72$).

Table 10 presents the mean program scores and time of the two treatments; Figure 37 and Figure 38 graphically present these data as box plots. Once again, while the Self Construction group scored higher, a two sample *t*-test found no significant difference between the two groups ($t = -1.067$, $df = 21$, $p < 0.300$). Similarly, while the Self Construction group took slightly less time on average, a two-sample *t*-test determined that the difference was not significant ($t = 1.083$, $df = 21$, $p < 0.29$).

Given that the Self Construction group performed better on the first two traces, while the Active Viewing group performed better on the second two traces, I performed a repeated measures test in order to determine whether treatment significantly affected trace task performance on a task-by-task basis; however, no significant effect was found (Wilk's Lambda = 0.722, $f_{3,19} = 2.443$, $p < 0.10$). Finally, a linear regression was performed in an attempt to correlate the four dependent measures with student grade point average. The analysis yielded $r^2$ values that ranged from 0.022 to 0.155. This suggests that grade point

average is an extremely poor predictor of performance, since it accounted for less than 16% of the variance in all cases.

Table 9. Mean Trace Scores and Times of the Two Treatments

| | Active Viewing | Self Construction |
|---|---|---|
| Average Case Trace (Perfect: 40) | | |
| Score (Std. Dev) | 22.34 (10.71) | 28.29 (8.99) |
| % Correct (Std. Dev.) | 55.7 (26.8) | 70.8 (18.9) |
| Time (Std. Dev.) | 18.55 (5.47) | 18.92 (5.44) |
| Best Case Trace (Perfect: 11) | | |
| Score (Std. Dev.) | 6.08 (1.64) | 7.09 (1.30) |
| % Correct (Std. Dev.) | 76.0 (20.5) | 88.6 (16.3) |
| Time (Std. Dev.) | 2.98[45] (1.17) | 4.01 (2.55) |
| Degenerate Case Trace (Perfect: 32) | | |
| Score (Std. Dev.) | 17.96 (9.97) | 17.83 (6.34) |
| % Correct (Std. Dev.) | 56.1 (31.1) | 55.7 (19.8) |
| Time (Std. Dev.) | 15.64 (5.62) | 15.54 (7.25) |
| Worst Case Trace (Perfect: 63) | | |
| Score (Std. Dev.) | 40.63 (17.75) | 36.76 (19.21) |
| % Correct (Std. Dev.) | 64.5 (28.2) | 58.3 (30.5) |
| Time (Std. Dev.) | 14.47 (3.67) | 15.38 (6.42) |
| Total (Perfect: 143) | | |
| Score (Std. Dev.) | 87.01 (35.68) | 89.97 (32.50) |
| % Correct (Std. Dev.) | 60.9 (25.0) | 62.9 (22.9) |
| Time (Std. Dev.) | 51.39 (13.46) | 53.86 (16.88) |

[45]There is one missing data point in the range of values used to calculate this mean.

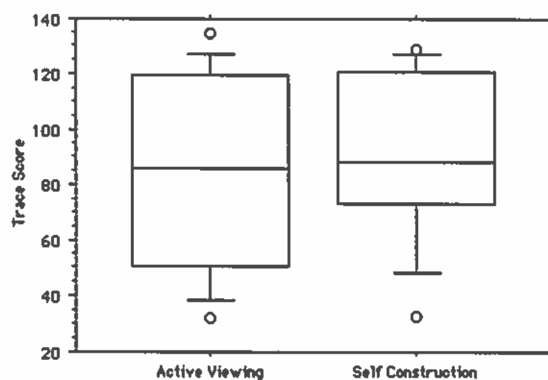



Figure 35. Box Plot of Trace Scores of Each Condition

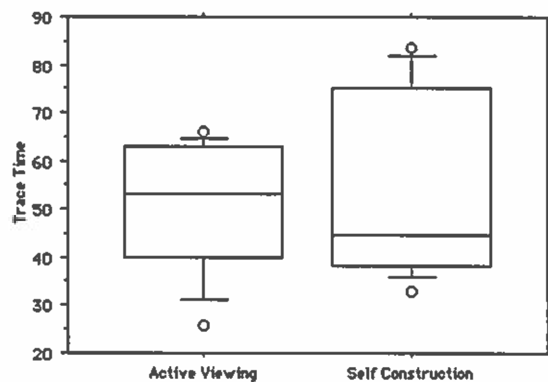



Figure 36. Box Plot of Trace Times of Each Condition

Table 10. Mean Program Scores and Times of the Two Treatments

| | Active Viewing | Self Construction |
|---|---|---|
| Program (Perfect: 31) | | |
| Score (Std. Dev.) | 15.58 (9.92) | 19.78 (9.32) |
| % Correct (Std. Dev) | 50.27 (32.01) | 63.81 (31.56) |
| Time (Std. Dev.) | 32.53 (4.32) | 30.32 (5.05) |

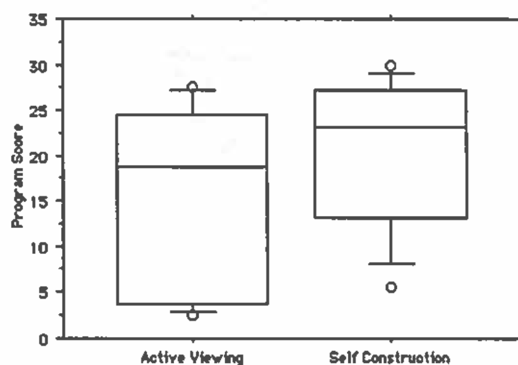



Figure 37. Box Plot of Programming Task Scores

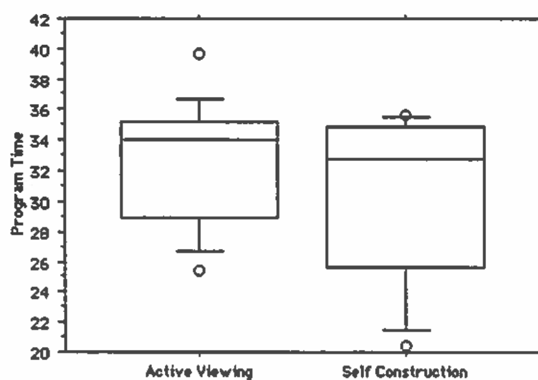



Figure 38. Box Plot of Programming Task Time

## Discussion

This experiment attempted to test rigorously the cognitive constructivist hypothesis that constructing one's own AV leads to better procedural understanding and recall than actively viewing an AV constructed by someone else. As has been the case for most of the researchers who have staged controlled experiments in an attempt to understand better the

pedagogical benefits of AV technology, I am faced with the difficult task of explaining why this experiment failed to confirm my hypothesis.

In interpreting the results, one should first double-check whether participants had any chance at all of doing well on the tasks. In attempt to avoid the kind of design flaw discovered in the experiment of Stasko, Badre, and Lewis (1993), I conducted an informal programming walkthrough (Bell, Rieman, & Lewis, 1991) prior to the study. My goal was to ensure that the information that participants would need to complete the two tasks would be available to them in the learning exercises in which they would engage. An important finding of that walkthrough, as discussed above, was that the learning materials originally lacked a display that made the important trace information more explicit. As a result, I added the "History" window to the predefined AV with which the Active Viewing group interacted, and I added the requirement that Self Construction participants construct their own history view. With respect to the programming task, my walkthrough noted that the three-page algorithm description that participants received included an explicit pseudocode description of the Select and Partition procedures. Since participants were likely to study this pseudocode as they explored the algorithm, and since participants had a background in programming, I expected that they would have a good chance of developing an ability to program the algorithm. If nothing else, they could elect to memorize the code—an approach taken by at least a couple of participants, according to exit questionnaire responses.

Departing from the procedure of prior experiments, I elected in this experiment to tell participants up front the kinds of things they would be expected to do in the evaluation phase of the experiment. I hoped that making the learning objectives explicit would enable participants to be more directed than they might have been if they had not known what would be expected of them.

A reality check of the data suggests that the experimental design succeeded in giving participants an honest chance of doing well on the tracing and programming tasks. On the tracing tasks, the highest individual percent correct was 94.41%, the lowest individual percent correct was 22.84%, and the average for all participants was 61.84%. On the programming task, the highest individual percent correct was 91.94%, the lowest individual percent correct was 8.06%, and the average percent correct for all participants was 56.24%. In both cases, it appears that the tasks were difficult, but not exceedingly difficult; participants appear to have had a chance to do well. Some participants excelled in the tasks, while others did poorly.

Abnormalities in the data might also explain the failure of this experiment to detect significant differences. Further inspection of the data does, in fact, reveal three pecularities.[46] First, one participant's time for the second trace is missing. Second, one participant's programming time exceeds the 35 minute maximum by over 4 minutes.[47] Third, the box plots presented above (Figure 35 – Figure 38) indicate that, while the trace scores and times are symmetrically distributed around the medium, the distributions of the programming scores and times are slightly skewed.[48] While these three peculiarities are worth noting, none of them is particularly serious, and none of them comes close to offsetting the overwhelming similarities observed between the Active Viewing and Self Construction data.

---

[46]See Appendix E for a table of all of the raw data on which the statistical analyses were performed.

[47]This is mysterious, given that I stopped all participants after 35 minutes. That several participants' programming times slightly exceed 35 minutes is completely expected, given that there were minor variations in the time they required to save the document.

[48]This might be explained by the fact that each trace score and time is actually the average of four data points, whereas each programming score and time is just one data point.

In fact, the means of the two treatments on all four measures appear so well matched that one might suspect that they come from identical populations. To test for the statistical equivalence of two treatments, Rogers, Howard, and Vessey (1993) introduce to social scientists a technique that is well established in the biological sciences. The test that they describe is essentially the converse of the t-tests used above: Rather than determining the probability that two samples could have come from the same population, a statistical equivalence test tells us the likelihood that two samples could be equivalent if they came from different populations. To determine whether the trace scores, trace times, program scores, and program times of the two treatment groups are statistically equivalent, I first made the assumption that no meaningful difference exists between scores and times that are within 5% of each other. Then, using the confidence interval approach described by Rogers, Howard, and Vessey (1993), I performed four statistical equivalence tests—one for each dependent measure. All four tests of equivalence failed, with p values that fall within a range (0.20 to 0.49) that is quite similar to the range of p values produced by the *t*-tests presented above.

Thus, it appears that the most plausible reason that no significant differences or similarities were detected between the two treatments is that the self-construction factor, as manipulated in this experiment, is simply not reliable enough to produce an effect. In other words, if one gives two groups of students two and a half hours to learn an algorithm using these two alternative approaches, neither group is likely to perform appreciably better on tracing and programming tasks. Indeed, the bleak reality is that, given such tremendous amount of variance in the data (which is quite common in studies of human cognitive performance), an experiment of this kind has little chance of demonstrating an effect. To see how small the chances for success really are, one need only perform a back-of-the-envelope power analysis (see, e.g., Cohen, 1988). Take, for example, participants' tracing task

accuracy. In order for the Self Construction group to have performed significantly higher on that task, it would have needed an average trace score of roughly 116—a full 26 points higher than it scored. Conversely, given the small difference (< 4%) between the mean tracing task scores of the two groups, one would require a sample size of far more than 1,000 participants per treatment group in order to have just a 50% chance of detecting such a small difference with statistical significance!

This is not to say that an experiment like the one reported here could not show an effect, if it were strong. Although the difference between the two group's average scores and times appears small, a visual inspection of the box plots presented above clearly indicates that there is less variance in the data of the Self Construction group. Given this, it seems plausible that applying the two treatments over a longer time could serve to solidify any differences between the two groups. For example, if students were to use these two alternative techniques to learn, say, five algorithms over the course of an entire academic quarter, one might see the slight trend in favor of the Self Construction group noted in this experiment turn into a significant difference. Unfortunately, because longer term studies like this are fraught with pragmatic difficulties, and can incur substantial financial overhead, they are beyond the scope of a single dissertation.

## Summary

Chapter V proposed a series of hypotheses inspired by the ethnographic fieldwork reported in Chapter IV. This chapter has proposed a series of three planned experiments for systematically exploring the effects of the three factors included in one of those hypotheses: the Understanding and Recall Hypothesis. Each experiment in the series manipulates one factor, while holding the others as equivalent as possible. Further, the results of previous

experiments can inform the design of subsequent experiments by suggesting the best values of the factors that have already been explored.

This chapter reported the results of the first experiment in the series, which I conducted within the scope of a later offering of the same CIS 315 course that my ethnographic fieldwork investigated. That experiment pitted the EF-minded pedagogical approach of having students interact with a pre-defined AV against the cognitive constructivist-minded pedagogical approach of having students construct their own AVs. The hypothesis was that the latter leads to better procedural understanding and recall than the former. Unfortunately, the results of the experiment fail to confirm that hypothesis. Since this failure does not appear to be due to design flaws in the experiment, one must take the result at face value: As manipulated in this experiment, the self construction factor does not produce an effect that is reliable enough to measure.

However, the results of this experiment would appear to have important practical implications. Using conventional (high epistemic fidelity) AV technology to prepare visualizations for classroom use is notoriously difficult (see, e.g., Brown & Sedgewick, 1985). In contrast, having students construct their own low epistemic fidelity visualizations using art supplies is cheap, easy, and requires a relatively small investment of time on the part of instructors. Thus, instructors looking for a low-overhead way to incorporate AV technology into their curricula can take comfort in the finding that students who use "low tech" art supplies to construct their own visualizations may learn algorithms just as well as students who interact with AVs developed by their instructor with conventional "high tech" AV technology.

Furthermore, given that the Recall and Understanding Hypothesis includes two other factors, it is plainly too early to reject the hypothesis altogether. In fact, in reflecting on the results of the experiment, one cannot help but speculate that the instructor

communication component of the hypothesis, in interaction with self-construction, may be vitally important in facilitating the improved understanding and recall posited by the Understanding and Recall Hypothesis. Indeed, it is interesting to note that, because they did not have the opportunity to receive feedback from an expert, participants in the Self Construction condition frequently constructed AVs that did not accurately portray the algorithm. On the other hand, participants in the Active Viewing group studied an AV that was guaranteed to always be correct. Given this fact, it is remarkable that the Active Viewing group did not significantly outperform the Self Construction group. One can only wonder how much better the Self Construction group might have done, had they been able to discuss their AVs with an expert. The next experiment in this series, which explores the effect of self construction in interaction with instructor communication, would thus appear to have a greater chance than this experiment of measuring a significant effect.

# CHAPTER VII

# PROTOTYPE LANGUAGE AND SYSTEM

> [One of computer science's] distinctive activit[ies] is building things, specifically computers and computer programs. Building things, like fieldwork and meditation and design, is a way of knowing that cannot be reduced to the reading and writing of books. To the contrary, it is an enterprise grounded in a routine daily practice. Sitting in the lab and working on. . .[computer] programs, it is an inescapable fact that some things can be built and others cannot. (Agre, 1997, p. 10)

The four hypotheses posited in Chapter V have important implications for the design of AV technology to be used as a pedagogical aid within the scope of an undergraduate algorithms course. Table 11 summarizes the specific design implications of each hypothesis. At first glance, an AV system built according to the design guidelines listed in Table 11 would differ in at least three fundamental ways from extant AV systems. First, whereas extant systems almost universally support the creation of input general, high typeset fidelity AVs, the guidelines recommend supporting the construction of low fidelity, input specific AVs. Second, in order to create such AVs, users of many extant systems are required to use quantitative, rather than direct, graphics. Finally, whereas extant AV systems almost universally support Brown's (1988) "playback" (start, stop, pause) interface for controlling AV execution (see Chapter II), and almost universally prohibit users from marking up or modifying an AV as it is executing, the guidelines advocate both flexible (i.e., forwards and backwards) execution control and support for dynamic mark-up and modification.

Table 11. Implications of Framework Hypotheses for AV Technology Design

| Hypothesis | Implied Guidelines For AV Technology Design |
| --- | --- |
| *The Activity Relevance Hypothesis*: Low input generality, low typeset fidelity, and direct graphics cause high activity relevance | 1. Support the creation of low typeset fidelity, one-shot AVs via direct manipulation, in order to promote engagement in relevant activities |
| *The Communication Effectiveness Hypothesis:* Low epistemic fidelity causes high communication effectiveness | 2. Support the creation of low typeset fidelity AVs, in order to promote high communication effectiveness |
| *The Understanding and Recall Hypothesis*: Self-constructing AVs with a story line, and then presenting them to an instructor for feedback and discussion, causes high recall and understanding | 3. Support *student* AV construction, in order to promote students' active construction of their own understanding<br>4. Support construction of AVs with a story line, in order to promote recall<br>5. Support easy AV mark-up and modification, as well as flexible control of AV execution, in order to support communication about algorithms |
| *The Community-Building Hypothesis*: Self-construction and high instructor communication cause high community-building | 6. Use AV technology as the basis for pedagogical exercises in which students construct their own AVs, and then present those AVs to their instructor for discussion, in order to promote increasingly expert participation in the community |

While these guidelines make specific recommendations regarding *what* tasks users should be able to accomplish with the system (*viz.*, the creation and presentation of low epistemic fidelity AVs), they offer relatively little guidance with respect to *how* users should accomplish those tasks. For example, the guideline that AV creation should be via direct manipulation constrains the design space, but stops short of specifying the precise user actions for accomplishing that task. In addition, notice that these guidelines fail to furnish the vitally important *conceptual model* for the system—that is, the underlying model that assists the user in understanding how to use the system. Thus, while these guidelines would appear to serve as a helpful starting point for the development of an alternative AV system—one that supports both the creation and presentation of low epistemic fidelity AVs—they stop short of furnishing a detailed design specification for the system.

In this chapter, I explore the design space circumscribed by the guidelines more fully by presenting an actual prototype AV system grounded in them. In so doing, I aim not only to demonstrate that a system rooted in the hypotheses is practical and feasible, but also to illustrate, as concretely as possible, what such a system might look like. The hope is that the prototype system will draw out the vivid contrasts between the design implications of EF Theory and the sociocultural constructivist-inspired theory being developed in this dissertation.

The foundation of the prototype is SALSA (Spatial Algorithmic Language for StoryboArding), an interpreted, high-level language for programming low epistemic fidelity AVs. Whereas conventional AV technology requires one to program a high epistemic fidelity AV by specifying explicit mappings between an underlying algorithm and the AV, SALSA enables one to specify a low epistemic fidelity AV that drives itself; the notion of an external "driver" algorithm is jettisoned altogether. In order to support AVs that drive themselves, SALSA enables the logic of an animation to be specified explicitly in terms of the *spatiality* of the AV—that is, in terms of the spatial relations (e.g., *above*, *right-of*, *in*) among objects in the AV.

The second key component of the prototype is ALVIS (ALgorithm VIsualization Storyboarder), an interactive, direct manipulation front-end interface for programming in SALSA. ALVIS strives to make constructing a SALSA animation (i.e., a "storyboard") as easy as, and ideally easier than, constructing a homemade animation out of simple art supplies. To do so, its conceptual model is firmly rooted in the physical metaphor of art supply storyboard construction, which, in previous empirical studies, has proved to be exceptionally natural, quick, and easy. Moreover, in contrast to existing AV technology, ALVIS also aims to support the interactive presentation of storyboards. To do so, its presentation interface goes beyond existing AV technology's standard "playback" (start,

pause, step) interface by supporting forwards and backwards stepping and execution, as well as dynamic mark up and modification.

In the remainder of this chapter, I present the design and implementation of ALVIS and SALSA in greater detail. I begin by deriving a specific set of design requirements for the prototype from actual empirical studies of humans constructing and presenting homemade animations. Next, I briefly describe the language and system by dissection. To make the descriptions more concrete, I then present an example in which the language and system are used to construct and present an actual (i.e., one observed in empirical studies) homemade animation of the insertion sort algorithm. Finally, having presented the prototype, I discuss its relationship to extant AV systems, as well as its implementation.

## Empirical Foundations

In surveying the literature on AV technology, one cannot help but notice a pattern: namely, designers' *intuitions* have most often been the guiding force behind its design. One might call this practice "armchair design." An important goal of the system-building effort presented in this chapter is to move beyond armchair design by grounding the design firmly in empirical data[49] as part of an iterative, *user-centered* design process (see, e.g., Norman & Draper, 1986). In this section, I make explicit the ways in which empirical data has informed the design of SALSA and ALVIS. Specifically, I present a corpus of relevant empirical data, from which I derive a set of high-level design requirements.

---

[49]In a sense, empirically-driven design is what this dissertation is all about. Indeed, the theoretical shift for which I am arguing emerged not by sitting in an armchair and imagining how AV technology might be pedagogically effective, but rather by using empirical data as a basis for understanding what algorithms students and instructors actually do with AV technology, and how it contributes to their understanding, identity, and competence.

The design requirements are based on three main sources of empirical data:

1. the two ethnographic studies presented in brief in Chapter IV, and described more fully in Appendices A and B;
2. past empirical studies of how humans construct algorithm animations from conventional art supplies (Chaabouni, 1996; Douglas, Hundhausen, & McKeown, 1995; 1996); and
3. an unpublished pilot study I conducted in which computer science students were asked to watch algorithm animations and then to describe them.

The pertinent observations from these empirical studies give insight not only into what the homemade AVs created by users of ALVIS and SALSA should contain, but also into how users should construct, execute, and present their homemade AVs. Below, I briefly summarize the key observations with respect to AV content, AV construction, and AV execution and presentation.

## AV Content

In my ethnographic fieldwork, I gathered well over 100 examples of AVs constructed by 300-level algorithms students. Given the finding that low epistemic fidelity AVs better support the objectives of a 300-level algorithms course than do high epistemic fidelity AVs, the forty *storyboards* that were constructed using conventional art supplies (including transparencies, pens, and paper) provide the most relevant design information. These storyboards depicted divide-and-conquer, greedy, dynamic programming, and graph algorithms, including Huffman compression, QuickSelect, breadth-first and depth-first search, and Dijkstra's shortest path algorithm. The following generalizations can be made regarding their content:

1. The storyboards consisted of groups of movable, labeled objects of arbitrary shape (but most often boxes, circles, and lines) arranged on a static background; regions of the display, and locations in the display, were also significant.
2. Objects in storyboards were frequently arranged according to one of three general layout disciplines: *grids*, *trees*, and *graphs*.
3. The most common kind of animation was simple movement from one point to another; pointing (with fingers) and highlighting (circling or changing color) were also common. Occasionally, multiple objects were animated concurrently—for example, two objects moving concurrently along the same path.
4. The majority of the storyboards unfolded within a single "window" (rectangular area); occasionally, multiple "windows" depicted multiple, synchronized views of an algorithm, or multiple views of alternative algorithms operating on the same data set (e.g., breadth-first vs. depth-first search).

## Process of AV Construction

Douglas, Hundhausen, and McKeown (1995; 1996) and Chaabouni (1996) used Interaction Analysis (Jordan & Henderson, 1995) to study in detail the process by which humans construct AVs using simple art supplies. Their observations suggest that AV storyboard designers create objects by simply sketching them out (or cutting them out) and placing them on the page. Design tends to be an experimental, dialectical process, rather than a calculated, linear one. For example, new objects are added to the storyboard as the need for them arises. Similarly, decisions regarding how to animate these objects tend to be made tentatively on the spot, not firmly in advance. Furthermore, the size of the objects, as well as their placement, is invariably done not in terms of absolute coordinates, but rather

*relative* to other objects that have already been placed in a storyboard; AV designers never place or move objects according to Cartesian coordinates.

In an unpublished pilot study in which I asked students to view and describe algorithm animations, I made three observations that complement the ones just presented. First, participants invariably gave animation objects descriptive names; however, two participants rarely chose the same name for the same object. Second, participants tended not to refer to animation objects using Cartesian coordinates. Instead, they referenced objects by using either descriptive labels, or by describing them in terms of their *spatial relationship* to other objects, which served as landmarks. Finally, participants tended not to supply exact destination locations of animation actions (e.g., *move object1 to 20,30*). Rather, destination locations were specified either (a) as an implicit location described in terms of its spatial relationship to other objects (e.g., *move object1 below object2* or *move object1 below object right-of object1*); or (b) as an implicit location described in terms of a direction with respect to an object's present location (e.g., *move object1 left*). In the case of (b), the distance that the object was to be moved often was not explicitly specified, but was usually the length or width (or fraction thereof) of an adjacent object.

## Process of AV Execution and Presentation

In this process, AV designers set their storyboard into motion to illustrate the execution of an algorithm on a particular data set. Observations made by Douglas, Hundhausen, and McKeown (1995, 1996) and Chaabouni (1996) indicate that, rather than referring to program source code or pseudocode, AV storyboard designers tend to simulate their storyboards by paying close attention to, and hence maintaining, important *spatial relations* among storyboard objects. For example, rather than maintaining a looping variable

and stopping a loop when that variable reaches a certain value, storyboard designers might instead observe whether an object representing that loop variable reaches a certain position in the storyboard. For example, they might stop looping when an object advances to the right of a row of boxes.

In the presentation sessions I observed in my fieldwork, students provided verbal play-by-play narration as they executed their storyboards. They used deictic gestures (both with their fingers and the tips of pens) extensively to coordinate their narration with objects in their storyboards. Frequently, audience members broke in with comments or questions. To clarify such comments and questions, audience members often pointed to objects in the storyboard, or even marked up the storyboard with a pen. In response to such comments and questions, student presenters paused their animation, or even fast-forwarded or rewound it to another point of interest. In some cases, audience suggestions led to on-the-spot modifications of the storyboard—for example, changing a color scheme, adding a label, or altering a set of input data.

## Design Requirements

Taking the above observations into account implies five high level functional requirements for SALSA and ALVIS:

1. *Storyboard content.* SALSA and ALVIS must be capable of expressing the AV storyboards that I observed in my ethnographic fieldwork.
2. *Storyboard creation process.* SALSA and ALVIS must enable users to express storyboards using the same dialectical, experimental process that empirical study participants adopted. In other words, the system encourage experimentation by supporting a short modify-compile-execute cycle.

3. *No Cartesian coordinates.* SALSA and ALVIS must enable users to express storyboards in the same terms that the study participants used—using spatial relations, not Cartesian coordinates.
4. *Spatial execution model.* SALSA and ALVIS must support an execution model similar to that adopted by empirical study participants—one based on *spatial*, rather than algorithmic logic.
5. *Interactive presentation.* SALSA and ALVIS must enable users to present their storyboards as part of interactive discussions. This entails an ability to pause and restart storyboards; to rewind and fast forward storyboards to points of interest; and to point to, mark-up, and modify storyboards as they are being presented.

In addition, two high level usability requirements are implied by the empirical data:

1. *Quick and Easy Storyboard Creation.* Undergraduate algorithms students should be able to create homemade AVs quickly and easily with SALSA and ALVIS—at least as quickly and easily as they can create storyboards using conventional art supplies, and ideally more quickly and easily.
2. *Fluid, Interactive Presentation.* Undergraduate algorithms students should be able to use SALSA and ALVIS to fluidly present their animations to an audience. In other words, in response to audience questions and comments, student presenters should be able to control their animations' execution easily, and to modify and mark up their animations easily; the interface should not get in the way of the presentation.

In the following two sections, I give brief tours of the two key components of the prototype technology designed to meet these requirements: SALSA and ALVIS.

## SALSA

The foundation of the prototype system is SALSA, a high-level, interpreted language for programming low epistemic fidelity AVs, or *storyboards*.[50] In line with the requirements identified above, SALSA scripts (a) produce low typeset fidelity storyboards (i.e., they resemble sketches, rather than textbook figures); (b) are devoid of Cartesian coordinates; are capable of expressing spatial relations and logic; and (c) operate on a specific set of input data, rather than general input.[51]

The remainder of this section describes SALSA by dissection, beginning with the data types, moving to the language's support for spatial relations, and concluding with a synopsis of the SALSA commands. A more detailed summary of the SALSA language can be found in Appendix F.

### Data Types

SALSA defines three data types, each of which models one of the core elements of the "art supply" storyboards observed in the empirical studies discussed above:

---

[50]The prototype versions of SALSA and ALVIS run on top of Microsoft® Windows 95. The SALSA interpreter was programmed in Microsoft Visual C++™ with the help of the Microsoft® Foundation Classes and the Cygwin port of Gnu Flex and Gnu Bison. The ALVIS interface, including its back-end animation engine, was implemented in Microsoft Visual C++™ using the Microsoft® Foundation Classes.

[51]This point requires further explanation. A given SALSA script operates on objects that represent one specific set of input data. However, the fact that SALSA is an interpreted language makes it easy to redefine the objects of the script that represent the input data. Hence, as long as the internal logic of a SALSA script is not "hard coded" for a specific set of input data, it is possible to redefine the objects of the script, and then to re-execute the script to observe the animation operating on an alternative set of input data. This kind of "pseudo input generality" will be illustrated in the example presented later in the chapter.

1. *Cutout.* This is a computer version of a construction paper cutout. It can be thought of as a movable scrap of construction paper of arbitrary shape on which graphics are sketched. Associated with the cutout data type are several attributes. Some of these are purely spatial, and define various x,y positions (called "refpoints") in and around the cutout (e.g., left-center, top, bottom; see Figure 39). Others have to do with the appearance of the object, such as *graphic-rep* (a file containing the graphics of the cutout), *visible*, and *highlighted.*
2. *Position.* The position data type represents an x,y position within a storyboard.
3. *S-struct.* As discussed in the previous section, empirical studies suggest that AV designers use the same spatial layout patterns time and time again. A *spatial structure* (*s-struct* for short) can be thought of as a closed spatial region in which a set of (not necessarily homogeneous) cutouts can be systematically arranged according to a particular spatial layout pattern. The prototype implementation of SALSA supports just one s-struct: *grids.*

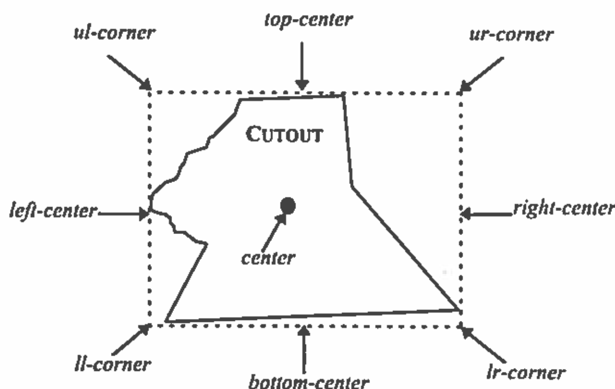



Figure 39. Predefined Cutout Refpoints

## Spatial Relations

As discussed in the previous section, a key requirement is that SALSA programmers not have to deal with Cartesian coordinates in any form. SALSA addresses this requirement by supporting *spatial relations*, which provide a mechanism both for describing storyboard elements in terms of how they relate spatially to other storyboard elements, and for expressing storyboard logic in terms of the spatiality of a storyboard.[52] Consider, for example, the sample storyboard in Figure 40. The viewer of this storyboard could make a great number of statements about the ways in which the various cut-outs in this storyboard are spatially related to each other. For instance, *square is left-of hexagon*; *square is right-of triangle*; *cross is above square*; *arrow is below square*; *diamond is above hexagon*; *diamond is touching hexagon*; *oval is in hexagon*; and *arrow is outside-of square.*

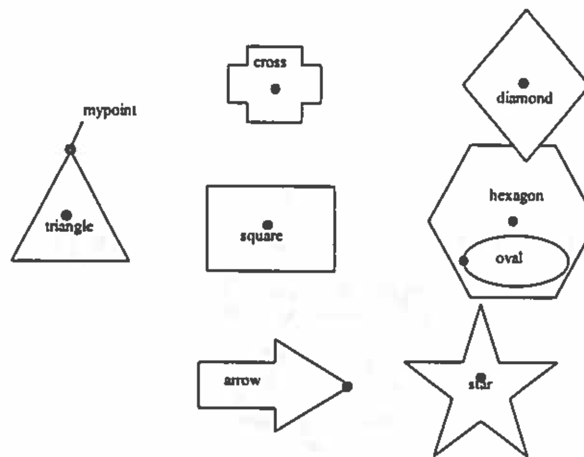



Figure 40. Cutouts in a Sample SALSA Storyboard

---

[52]For an empirically-based account of how humans use spatial relations, see (Douglas, Novick, & Tomlin, 1987).

One might wonder how *diamond* can be both *above*, and *touching*, *hexagon*. The answer is that each cutout has a distinct *reference point* that is implicitly referenced in any spatial relations statement. For the purpose of illustration, each cut-out's implicit reference point is indicated by a dot in Figure 40. In SALSA, one may set a cut-out's *refpoint* attribute to any of the nine different locations shown in Figure 39.

The concept of a *view plane* adds a third dimension to spatial relations in SALSA. In Figure 40, notice that the diamond is actually closer to us than the hexagon, since it obscures a portion of the hexagon. In SALSA, each cut-out occupies a distinct view plane, making it possible to express spatial relations that involve a cut-out's "closeness" to us (the viewers)— for example, *hexagon is further-away-than diamond*, and, conversely, *diamond is closer-than hexagon*. Finally, notice that Figure 40 includes a *position* called *mypoint*. It is also possible, in SALSA, to state and test whether two points are *at* the same location. With respect to the sample storyboard in Figure 40, one could say *top-center of triangle is at mypoint*.

In sum, Table 12 lists SALSA's 10 spatial relations keywords, which enable one to express any number of spatial relations among the elements of a storyboard.

Table 12. The Ten Spatial Relations Supported By SALSA

| | | | |
|---|---|---|---|
| *above* | *below* | *left-of* | *right-of* |
| *touching* | *in* | *outside-of* | *closer-than* |
| *further-away-than* | *at* | | |

## Commands

Table x briefly summarizes the 12 commands defined by SALSA. As the table indicates, these commands fall into three main categories. The first category of

commands supports storyboard element creation, deletion, and attribute modification; the second supports conditional branching and iteration; and the third supports various forms of animation.

Table 13. Brief Summary of SALSA Commands

| Command | Description |
|---|---|
| *Create* | Creates a new cutout, position, or s-struct. The *Create as clone* variant enables one to create a new object as a copy of an existing object. |
| *Place* | Actually positions a created storyboard element in the storyboard. |
| *Delete* | Deletes a created storyboard element. |
| *Assign* | Creates an alternative identifier for a created storyboard element. |
| *Set* | Sets an attribute of a storyboard element to a new value. |
| *If-then-else* | Supports conditional branching. In most cases, the conditional will test spatial relations of storyboard elements, as described above. |
| *While* | Supports conditional iteration. |
| *For-each* | Supports iteration over (a) a set of positions associated with a grid or cutout, or (b) a set of storyboard elements that satisfy a boolean test. |
| *Move* | Animates a cutout from its present location to a new location over a specified length of time, and along a specified path. |
| *Resize* | Changes the size of a cutout over a specified duration of time. |
| *Flash* | Flashes a cutout over a specified period of time. |
| *DoConcurrent* | Enables a block of animation commands to be executed concurrently. |

## ALVIS

As the graphical front-end to the SALSA language, ALVIS provides both a direct-manipulation interface for constructing SALSA storyboards, and an interactive environment for executing and presenting them; it is thus the component of the prototype with which users directly interact. The conceptual model underlying ALVIS's storyboard construction interface is rooted in the physical metaphor of art supply storyboard construction. An important component of this metaphor is the concept of cutouts (or "patches"; see van de Kant, Wilson, Bekker, Johnson, & Johnson, 1998): scraps of virtual construction paper that may be cut out and drawn on, just like real construction paper. ALVIS users create

homemade animations by cutting out (with a scissors tool), sketching on (with a pen tool), and arranging (via direct manipulation) cutouts on a static background. They then specify, either by direct manipulation or by directly typing in SALSA commands, the ways in which the cutouts should be animated over time.

Likewise, the conceptual model underlying ALVIS's storyboard presentation interface is based on the physical metaphor of presenting an "art supply" storyboard. The interface supports four distinct features that are taken for granted in "art supply" presentations, but that are notably absent in conventional AV technology. First, it is possible in ALVIS to reverse the direction of storyboard execution in response to audience questions and comments. Second, ALVIS provides a conspicuous "presentation pointer" with which the presenter and audience members, may point to objects in the storyboard as it is executing. Third, ALVIS includes a "mark up pen" with which the presenter and audience members may dynamically annotate the storyboard as it is executing. Finally, presenters and audience members may dynamically modify a storyboard as it is executing by simply inserting SALSA commands at the current *insertion point* in the script—that is, the point in the script where execution is paused.

The remainder of this section gives a high level tour of the ALVIS interface. This will set the stage for the section that follows, in which I illustrate, by way of a detailed example, how ALVIS can be used to create and present a storyboard.

## Overview

Figure 41 presents a snapshot of the ALVIS environment, which consists of three main regions:

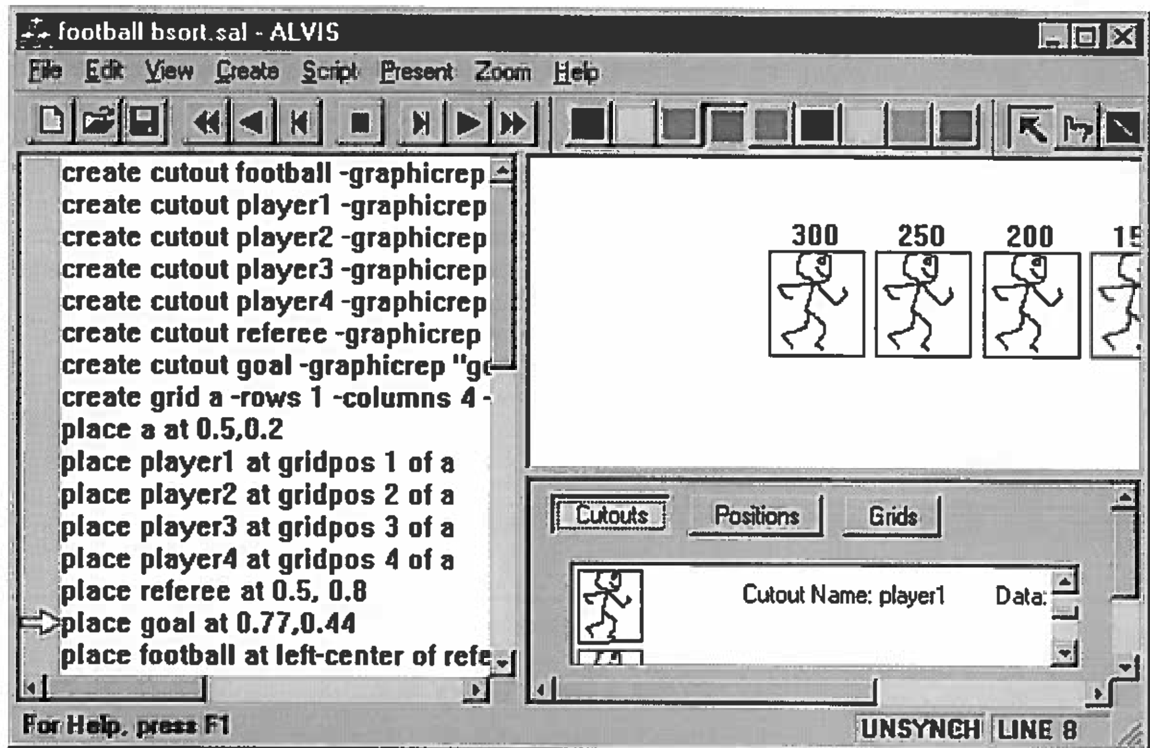



Figure 41. Snapshot of the ALVIS Environment

1. *SALSA Script view* (left). This view displays the SALSA script presently being edited; the arrow on the left-hand side of the view denotes the line at which the script is presently halted—the current "insertion point" for editing.
2. *Storyboard view* (upper right). This view displays the (graphical) storyboard generated by the SALSA script displayed in the *SALSA Script* view. The *Storyboard view* is always synchronized with the *SALSA Script* view. In other words, it always reflects the execution of the SALSA script up to the current insertion point marked by the arrow.
3. *Created Objects palette* (lower right). This area contains an icon representing each *cutout*, *position*, and *grid* that has been created thus far. When an icon is double-clicked, a dialog box containing the object's current attribute values appears. The *Created Objects* palette thus provides a convenient means of accessing and editing the attributes associated with each object. Note that, like the *Storyboard* view, this palette is

synchronized with the *SALSA Script* view. In other words, it displays all storyboard objects that have been created as a result of the script executing up to the current insertion point.

Users may access all of ALVIS's functionality through its menus, which are presented in Figure 42. The *File* menu contains standard commands for creating, opening, closing, saving, and printing storyboard (*.sal*) files, which are nothing more than plain text files containing a sequence of SALSA commands.[53] Similarly, the *Edit* menu comprises standard editing commands, all of which make sense when users are working within the SALSA Script View, but only some of which make sense at other times (e.g., *Delete* is available when an icon in the *Created Objects palette* is selected). The *View* menu toggles the visibility of all non-menu interface elements. The toolbars and *Cutout Graphics Editor* that appear on the *View* menu, along with all of the items on the *Create*, *Script*, and *Present* menus, assist users in storyboard creation and presentation; I describe them further in the two subsections on those topics that follow. Finally, the *Zoom* menu enables the user to set the zoom factor for the storyboard view; the higher the zoom factor, the larger the cutouts appear.

---

[53]Note that, in the prototype implementation, users may work with just one storyboard at a time. Note also that each *create cutout* command within a SALSA script contains a reference (i.e., a path and file name) to a .cut file containing the cutout's graphics. (The process by which users create such files will be discussed below.) Thus, in order for the script to run properly, all cutout graphics files referenced in the script must be readable.

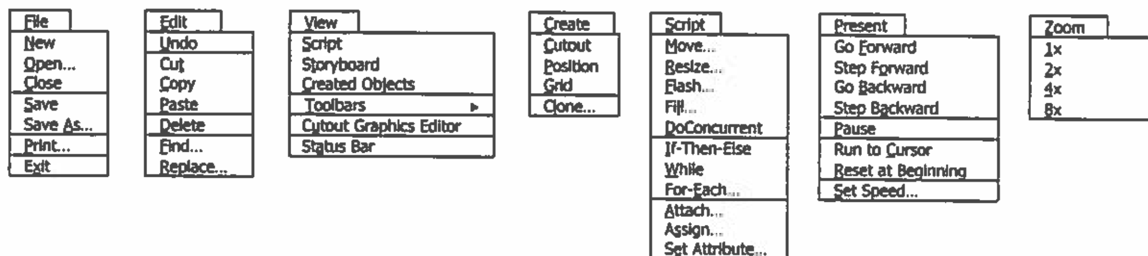



Figure 42. The ALVIS Menus

## Storyboard Construction Interface

In ALVIS, constructing a storyboard amounts to programming a SALSA script. At any point, ALVIS users always have at least two different ways of doing that:

1. Type SALSA commands directly into the *SALSA Script* view at the current insertion point.
2. Use the ALVIS interface to generate SALSA commands by directly manipulating objects in the *Storyboard* view and/or filling in dialog boxes; any commands so generated are automatically inserted in the *SALSA Script* view at the current insertion point.[54]

The two core subtasks in storyboard creation are (a) creating and placing storyboard elements, and (b) animating cutouts. With the proviso that these tasks may be accomplished by simply typing appropriate SALSA commands into the *SALSA Script* view, I now briefly discuss how the ALVIS interface supports them.

---

[54]It is important to note that this is an option only when the user performs edits at the current insertion point. If the user attempts to perform edits at some other point in the script, , the direct manipulation editing commands are disabled, since the *Storyboard* view does not reflect the current state of the SALSA script

Creating and Placing Storyboard Elements

By selecting items on the *Create* menu and filling in corresponding dialog boxes, users may generate appropriate "create" commands and insert them at the current insertion point in the script. The dialog boxes that appear in response to choosing items on the *Create* menu present the default values of the attributes for the object (*cutout*, *position*, or *grid*), and invite users to override those defaults.

For example, Figure 43 presents the dialog box that appears in response to selecting *Cutout*...on the *Create* menu. Users are invited to name the new cutout, but they need not; upon creation, each SALSA object is automatically assigned a unique ID, which may be used in lieu of a name to reference the object. Users specify a cutout's graphical by either selecting an existing .cut file (with the "Browse" button), or by creating a new .cut file. The user may create a new graphical representation for a cutout with the *Cutout Graphics Editor*, which appears in response to clicking the "Create New..." button in Figure 43, and is also accessible at any point through the *View* menu.

Figure 44 presents a snapshot of the *Cutout Graphics* editor. As in "art supply" storyboard construction, creating a cutout involves first using the scissors tool to cut out a virtual scrap of paper from a base shape of the user's choice (square and circle are presently supported), and then using the pen tool to sketch graphics onto the scrap of paper. At any point, pen color may be altered, and sketched graphics may be erased with the eraser tool. Once satisfied, the user may save the graphics to a .cut file; the file is then automatically inserted into the "Source File" field of the *Create Cutout* dialog box, and is available for use as any other cutout's graphical representation.

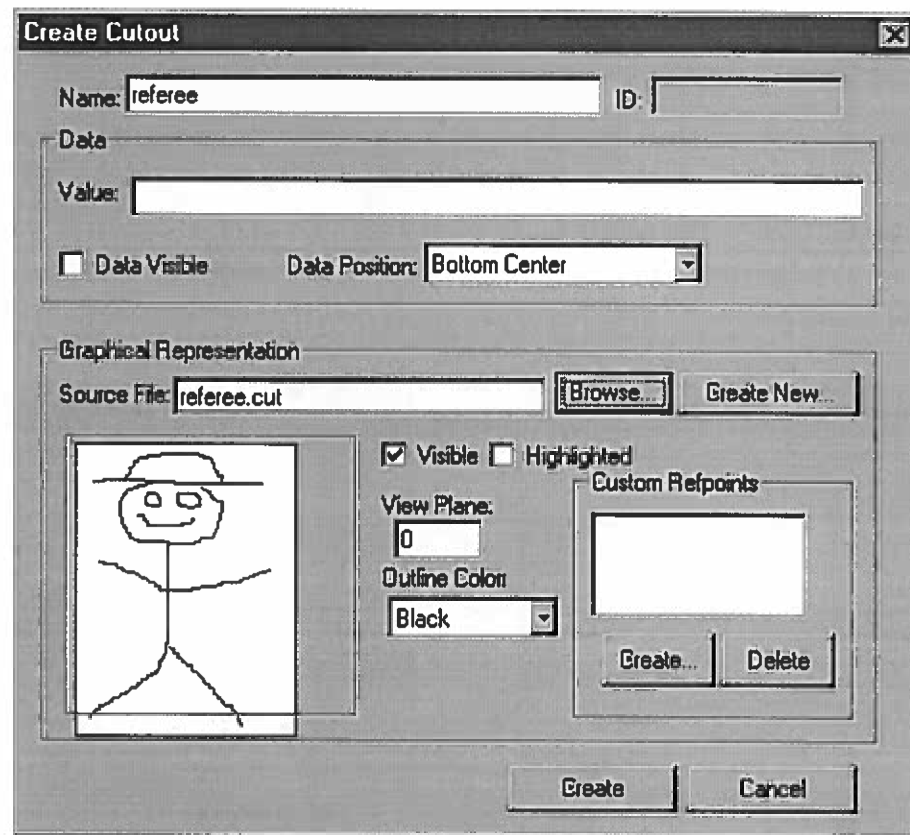



Figure 43. The Create Cutout Dialog Box

Recall that in SALSA, creating an object and placing an object are two different commands. Accordingly, in ALVIS, an object that is created through one of the *Create* dialog boxes does not immediately appear in the *storyboard* view; rather, it must be explicitly placed there. To do this, the user must drag-and-drop the newly created object from the *Created Objects* palette (where it appears immediately after creation) to a desired position in the *Storyboard* view; in response, the object appears in the *Storyboard* view, and ALVIS inserts an appropriate *place* statement into the script at the current insertion point.

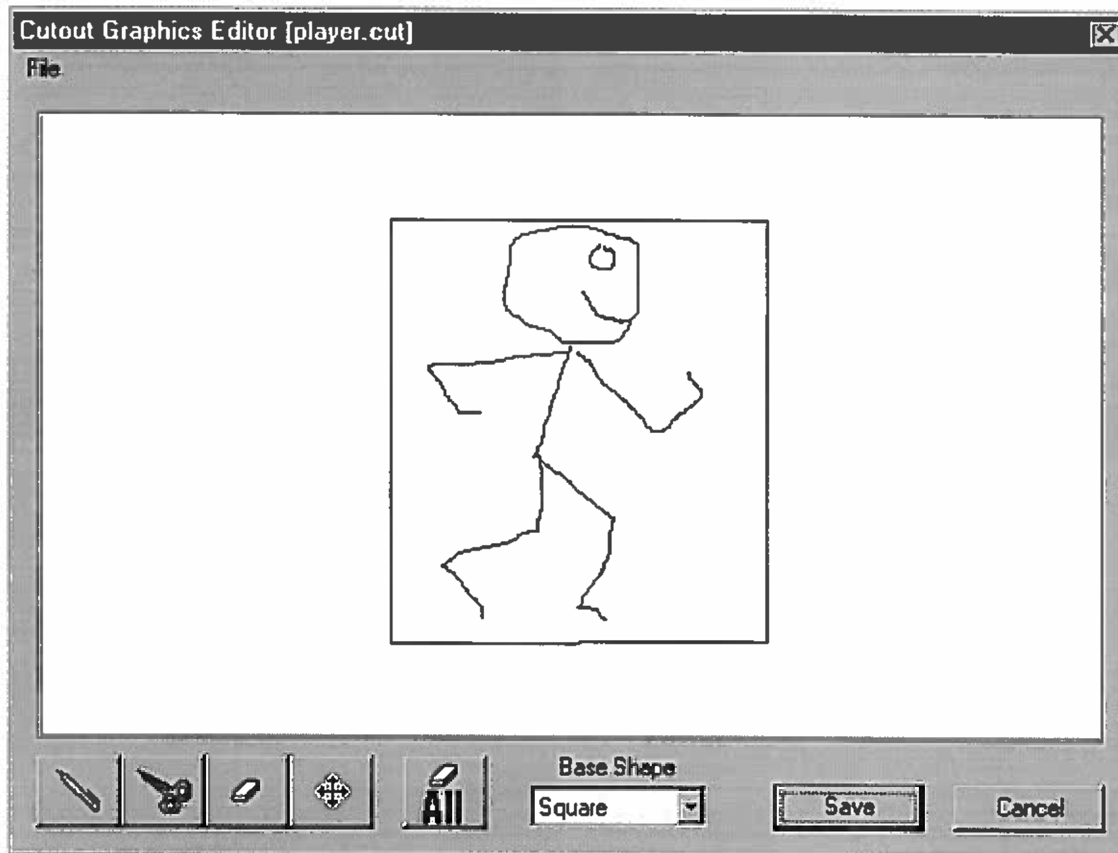



Figure 44. The Cutout Graphics Editor

Animating Cutouts

Recall that SALSA supports three different forms of animation—*move*, *resize*, and *flash*. In order to specify these forms of animation in ALVIS, users must first select the cutout to be animated in the *Storyboard* view, and then choose an animation command from the *Script* menu. In the case of *Move* and *Resize*, users are then asked to demonstrate the desired animation by directly manipulating the selected cutout—either by moving it along the desired path to the new location, or by resizing the cutout to the desired size by

manipulating the resize handles that appear. In all three cases, a dialog box pops up that invites users to refine the animation further.

For example, Figure 45 presents the *Script Move* dialog box. Using the move demonstrated by the user, ALVIS makes assumptions about the format of the SALSA command that will ultimately be generated, and inserts those assumptions into this dialog box. First, the duration of the demonstrated move is timed, and that time is inserted into the Duration field. Second, ALVIS assumes that the movement should be to an absolute position, and it uses the landing point of the demonstrated move as a basis for filling in the x and y coordinates. Finally, ALVIS assumes a custom path, and generates a series of offsets for that path based on the demonstrated move.

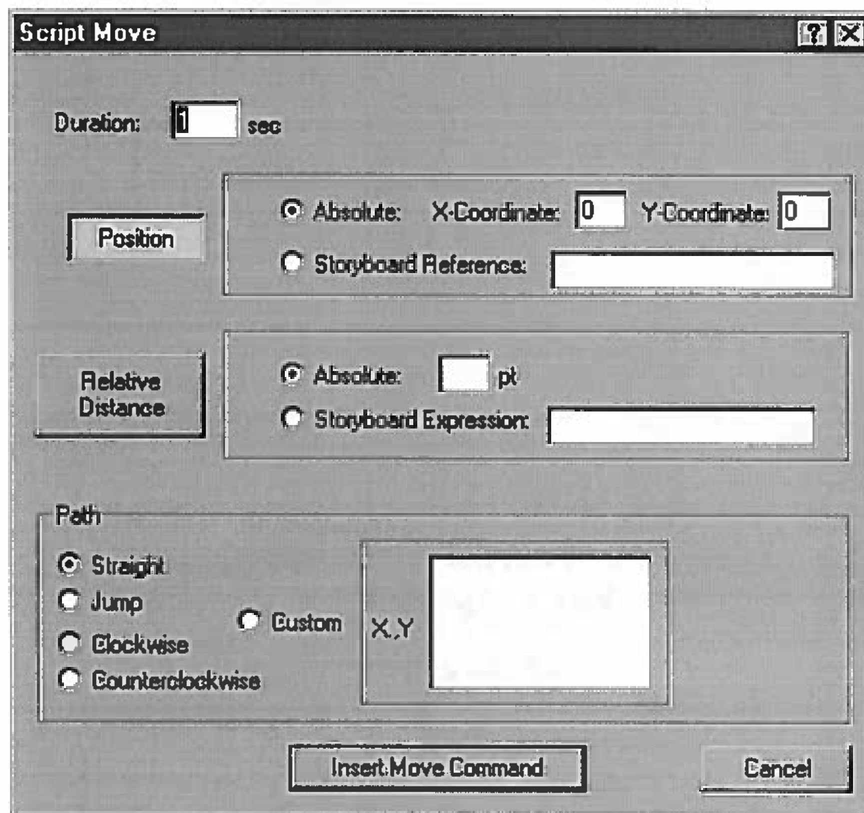



Figure 45. The Script Move Dialog Box

Clearly, most of these assumptions are naïve. Fortunately, the *Script Move* dialog box enables the user to correct the assumptions. For example, the user may have intended the movement to end at a particular storyboard reference point (e.g., *left-center of foo*). In that case, the user may override the default by clicking on the "Storyboard Reference" radio button and specifying a storyboard reference point in the corresponding text box. Likewise, the user may have actually wanted to drag a perfectly straight path. The user may override the default custom path in that case simply by selecting the "Straight" (Path) radio button.

## Storyboard Presentation Interface

Because the *SALSA Script* and *Storyboard* views are tightly synchronized, the tasks of storyboard creation and presentation are virtually inseparable in ALVIS. Indeed, in creating storyboard elements and specifying how they should be animated, ALVIS users necessarily execute their storyboard; there is no separate "presentation mode." In this way, ALVIS supports the dynamic modification requirement outlined above: At any point in a presentation, users may dynamically alter their storyboard simply by inserting or removing commands at the current insertion point.

In addition to dynamic modification, ALVIS supports several features that are specifically designed to assist in storyboard presentation. All of these tools are available on floating toolbars. The *Execution Control* toolbar (see Figure 46), whose functionality is fully duplicated by the items on the *Present* menu, provides users with an interface for executing their storyboards. This interface supports both forwards and backwards single-stepping and execution. Moreover, it provides a means of executing to an arbitrary point in the script, as marked by the current location of the cursor in the *SALSA Script* view.

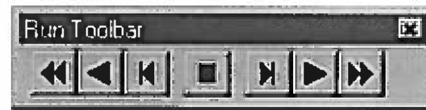



Figure 46. The Execution Control Toolbar

In discussions up to this point, I have taken it for granted that the first item on the *Present/Edit* toolbar (see Figure 47) , the "selection" tool, is active. During storyboard creation, it enables users to select objects in the storyboard and apply animation commands to them. However, the next four tools on the toolbar are geared toward presentation. When selected, the "presentation pointer" changes the cursor into a conspicuous pointer, while the "mark-up pen" changes the cursor into a virtual marking pen, whose color and thickness may be selected from the palette of colors on the *Pen Color* toolbar . To erase specific annotations, users may select the "eraser" tool; to wipe all annotations clean, users may select the "erase all" tool.

Figure 47. The Present/Edit Toolbar

## Example of the Prototype in Use

To make the features of SALSA and ALVIS just described more concrete, I now walk through an example in which I use ALVIS to create and present a simple storyboard: the "football" animation of the bubble sort algorithm observed by Douglas, Hundhausen, and McKeown (1995; 1996). Recall that the bubble sort, an $n^2$ algorithm, successively compares

adjacent elements, swapping them if they are out of order (see Figure 48). The result is that smaller elements "bubble up" to the front of the array, or, conversely, that heavier elements "sink" to the end of the array. Specifically, after the $n^{th}$ pass of the algorithm's outer loop (lines 2 to 8), we are guaranteed that the last $n$ elements of the array are in place.

```
1: BubbleSort(A,n)
2: for j ← n-1 to 1
3:   for i ← 1 to j
4:     if (a[i] > a[i+1])
5:       exchange a[i] ↔ a[i+1]
6:     end if
7:   end for
8: end for
9: end BubbleSort
```

Figure 48. Pseudocode Description of the Bubble Sort Algorithm

The "football" animation (see Figure 49) depicts the bubble sort algorithm by telling the story of a game of American football. Elements to be sorted are represented as football players whose varying weights represent element magnitudes. At the beginning of the game, the players are lined up in a row. The referee then tosses the ball to the left-most player in the line, who becomes the ball carrier. The object of the ball carrier is to "score" by advancing the ball to the end of the line. If the ball carrier is heavier than the player next in line, then the ball carrier simply tackles the next player in line, thereby switching places with him. If, on the other hand, the ball carrier is lighter than the player next in line, then the ball carrier is stopped in his tracks, fumbling the ball to the next player in line. This process of ball advancement continues until the ball reaches the end of the line, at which point the player at the end of the line tosses the ball back to the referee. A pass of the algorithm's outer loop thus completes. If, at this point, there are still players out of order, the referee tosses the ball to the first player in line, and another next pass of the algorithm commences.

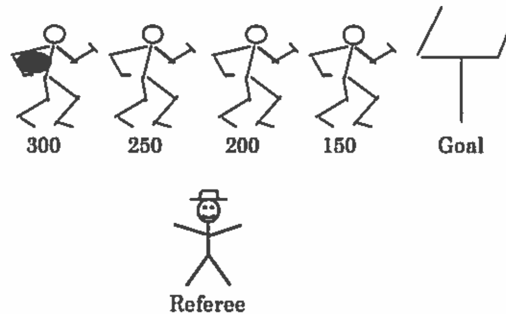



Figure 49. The Football Bubble Sort Storyboard

## Creating the Storyboard Elements

Having launched the ALVIS environment, I begin by creating the cutouts that appear in the animation of Figure 49: four players, a football, a referee, and a goal post. With respect to the football players, my strategy is to save time by creating one "prototype" player, and then cloning that player to create the other three players. This will work because I will opt to convey player weight not by varying player height or width, but rather by labeling each player, just as in Figure 49. To create the prototype player, I select *Cutout...* from the *Create* menu, which brings up the *Create Cutout* dialog box. I name the cutout "player1," and use the *Cutout Graphics Editor* to create its graphics in the file "player.cut": a square scrap of construction paper with a football player stick figure sketched on it. In addition, I set the data attribute to "300," and decide to accept all other default attributes.

When the *Create Cutout* dialog box is dismissed, ALVIS inserts the following SALSA create statement into the *SALSA Script* view:

*create cutout player1 –graphic-rep "player.cut" –data "300"*

In addition, the "player1" cutout appears in the *Created Objects* palette, and the execution arrow (in the *SALSA Script view*) is advanced to the next line, indicating that the create statement has been executed. Figure 50 shows a snapshot of ALVIS at this point.

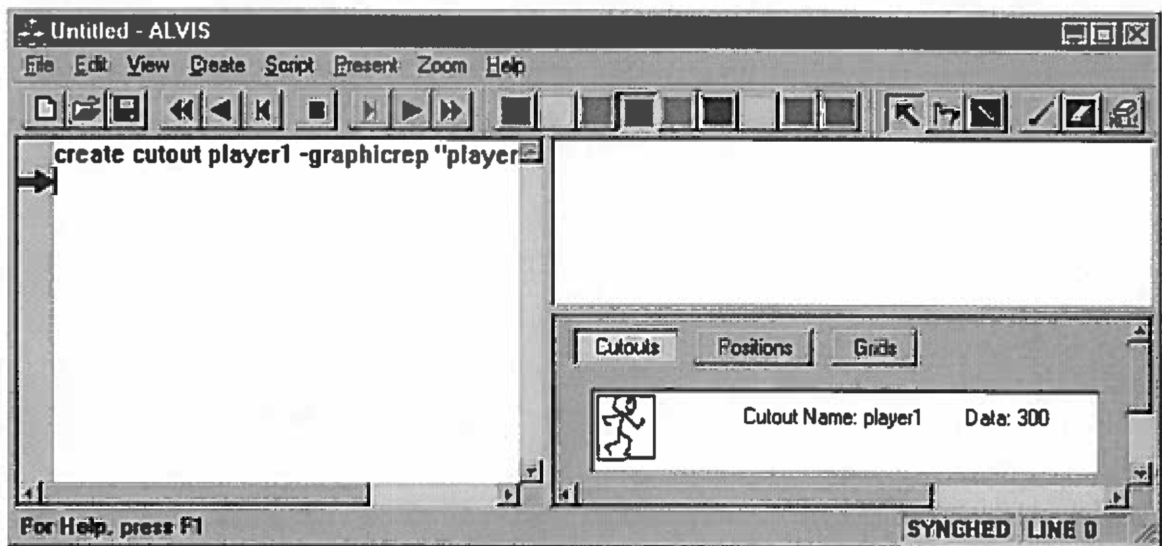



Figure 50. The ALVIS Environment After Creating First Player

I now proceed to clone "player1" three times. For each cloning, I first select the "player1" icon in the *Created Objects* palette, and then choose *Clone...* from the *Create* menu. This causes the *Create Cutout* dialog box to appear, with all attribute settings matching those of "player1." In each case, I change the name (to "player2", "player3", and "player4", respectively), and I change the data (to "250", "200", and "150", respectively), while leaving all other attributes alone. After dismissing the *Create Cutout* dialog box each time, a new *create* statement appears in the script—for example,

*create cutout player2 as clone of player1 –data "250"*

In addition, the new cutout appears in the *Created Objects* palette, and the execution arrow advances to the next line. I proceed to create the football, referee and goal post in the same way.

## Placing the Storyboard Elements

The next step is to place the cutouts in the storyboard. In order to lay the players out in a row, I use a simple *grid* s-struct, which I create by choosing *Grid*...from the *Create* menu and then filling in the *Create Grid* dialog box. I name the grid "a," so that it corresponds with the name of the array in the pseudocode of Figure 48. The grid needs one row and four columns, with a cell height and width corresponding to the height and width of "player." I accept all other default attributes, and click on the "Create" button to create the grid. Just before it is dismissed, the *Create Grid* dialog box appears as in Figure 51.

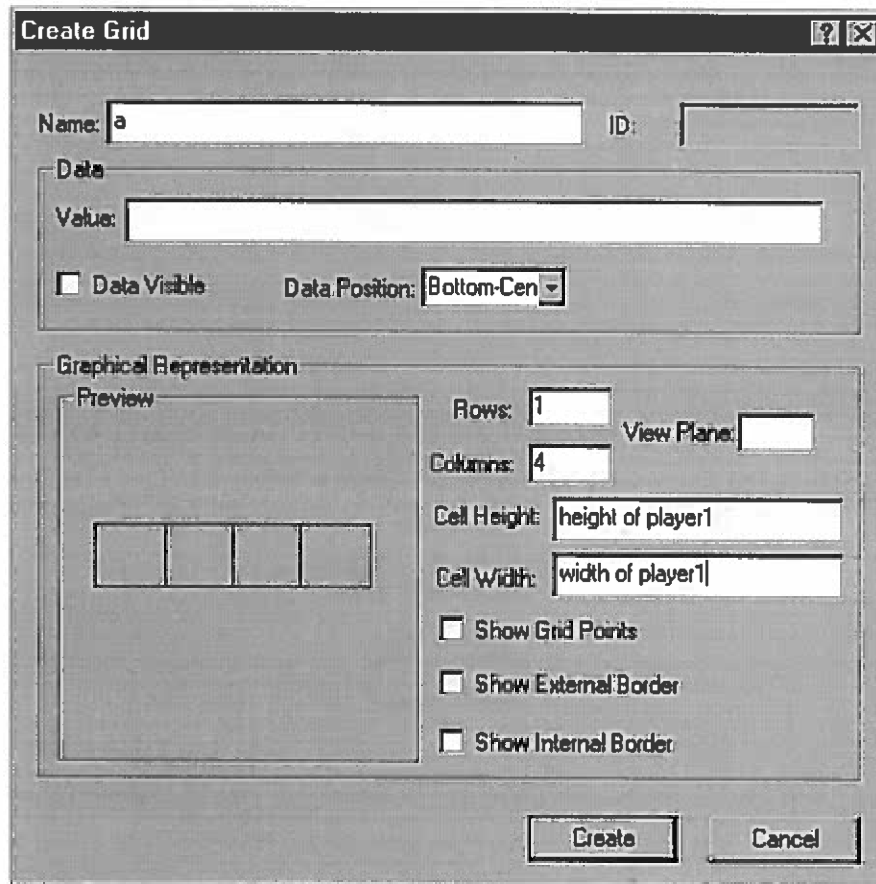



Figure 51. Creating a Grid with the Create Grid Dialog Box

I now proceed to place in the storyboard all of the objects I have created. First, I drag and drop the grid to an acceptable location in the middle of the storyboard; the corresponding *place* statement appears in the script, with the execution arrow advancing to the next line. With the grid in place, it is straightforward to position the players at the grid points:

*place player1 at position 1 of a*
*place player2 at position 2 of a*
*place player3 at position 3 of a*
*place player4 at position 4 of a*

Finally, I drag and drop the referee to a reasonable place below the row of football players, and I drag and drop the football to a place just above the referee. Figure 52 shows the ALVIS environment after all elements have been placed.

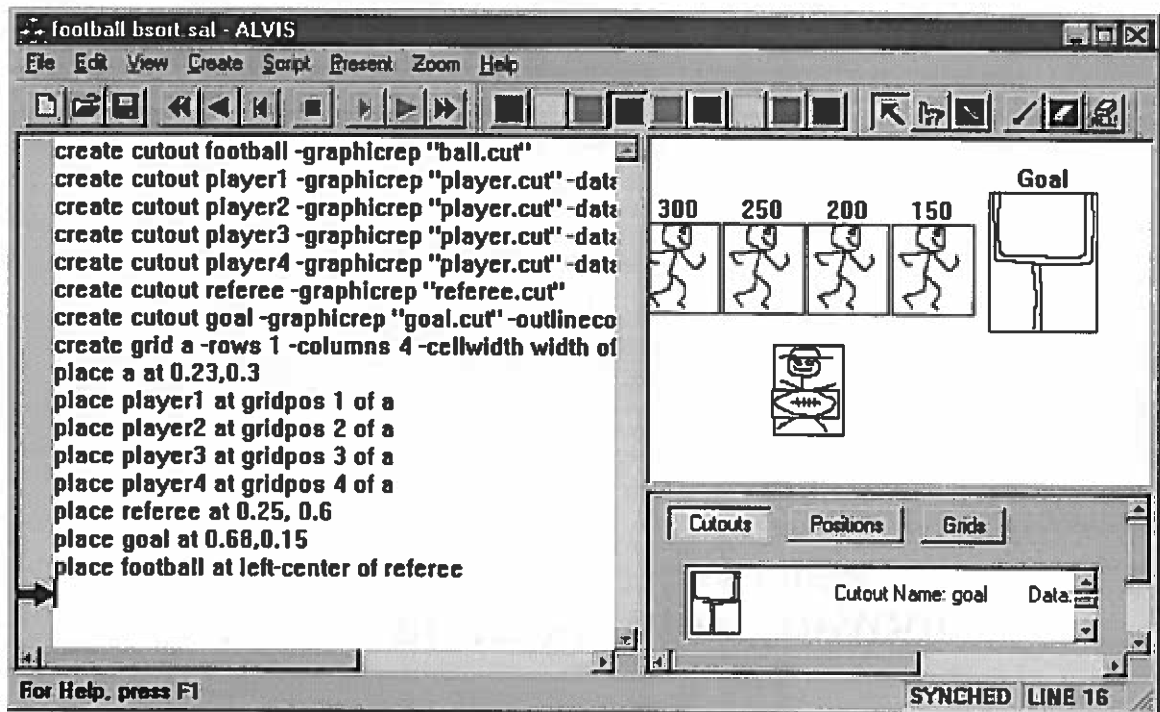



Figure 52. The ALVIS Environment After All Cutouts Have Been Placed

## Programming the Spatial Logic

I am now set to do the real SALSA programming necessary to make the storyboard work. From this point on, the SALSA code is of more interest than the ALVIS interface for programming it. Hence, I will focus mainly on the code itself in the discussion that follows, mentioning the ALVIS interface only in occasional footnotes.

As each player reaches his rightful place in the line, I want to turn his outline color to red. Since I have set the outline color of the goal post (the right-most cutout in the

storyboard) to red, we know that when the outline color of the cutout immediately to the right of the ball carrier is red, the ball carrier has reached the end of the line, and we are done with a pass of the inner loop. In SALSA, we may formulate this as a while loop:[55]

```
while outlinecolor of cutout right-of cutout touching football
    is red
        --do inner loop of bubblesort (see below)
endwhile
```

This code segment raises an important issue with respect to object resolution in SALSA; it is worth discussing briefly before I move on to code more of the storyboard. Note that the segment *cutout right-of cutout touching football* is potentially ambiguous; there may well be several cutouts that are touching the football, and there may well be several cutouts right of those. To resolve such ambiguous references, SALSA adopts a straightforward approach: Always take the best possible match. Say, for example, that more than one cutout were touching *football*. Then *cutout touching football* would resolve to that cutout which had the most area in common with *football*. Now assume that *cutout touching football* resolves to *player1*, and suppose that more than one cutout were to the right of *player1*. In that case, *cutout right-of player1* would resolve to the cutout that (a) is right of *player1* and (b) is the shortest Euclidean distance from *player1*. In sum, although SALSA supports ambiguous object references, it is important to keep in mind that ambiguous references like the one above may not always resolve to a unique cutout. In fact, in some cases, they may not resolve to any cutout at all.

Getting back to the football storyboard, we know that we are done with all passes of the outer loop when all but the first player in line has a red outline color. (There is no need

---

[55]To program a *While* loop through ALVIS, I could choose *While...* from the *Script* menu, which causes a while loop template to be inserted at the current insertion point of the script. Obviously, ALVIS is unable to program the loop for me!

to compare the first player to the second player, if the second player is known to be in order.) In SALSA, we may express this logic as another while loop:

```
while outlinecolor of cutout at position 2 of a is not red
        --do outer loop of bubblesort
endwhile
```

Now we come to the trickiest part of the script: the logic of the inner loop. We want to successively compare the ball carrier to the player to his immediate right. If these two players are out of order, we want to swap them, with the ball carrier maintaining the ball; otherwise, we want the ball carrier to fumble the ball to the player to his immediate right.

The following code makes the comparison, either fumbling or swapping, depending on the outcome:[56]

```
if data of cutout touching football >
   data of cutout right-of cutout touching football
    --swap players
    assign p1 to position in a of cutout touching football
    assign p2 to position in a of cutout right-of cutout touching football
    doconcurrent
        move cutout touching football to p2
        move cutout right-of cutout touching football to p1
        move football right cellwidth of a
    enddoconcurrent
else --fumble ball to next player in line
    move football right cellwidth of a
endif
```

With the inner loop complete, all that remains is to piece together the outer loop. At the beginning of the outer loop, we want to toss the ball to the first player in line. We then

---

[56]To insert an *if-then-else* template at the current insertion point, I could choose *If-then-else* from the *Script* menu. I could specify the *attach* statement by choosing *Attach...* from the *Script* menu, and then clicking, in turn, on the dependent and independent cutouts in the attachment. I could specify the *assign* statements by choosing *Assign...* from the *Script* menu, and then filling in the *Assign* dialog box. To program a *DoConcurrent* statement through ALVIS, I could choose *DoConcurrent...* from the *Script* menu, which inserts a *DoConcurrent* template at the current script insertion point. In ALVIS, I can program the flashes by first selecting, one at a time, the cutoouts to be flashed, and then selecting *Flash...* from the *Script* menu and filling in the *Flash Cutout* dialog box that comes up.

want to proceed with the inner loop, at the end of which we first set the outline of the player with the ball to red (signifying that he has reached his rightful place), and then toss the ball back to the referee. Thus, the outer loop appears as follows:[57]

*move football to left-center of cutout at position 1 of a*
*//inner loop goes here*
*set outlinecolor of cutout touching football to red*
*move football to top-center of referee*

Just one small detail remains: When the outer loop falls through, we might want to turn the outline of the first player in line to red, signifying that we are done sorting:

*set outlinecolor of cutout at position 1 of a to red*

Figure 53 summarizes this example by presenting the complete script.

---

[57]What if we wanted to generating this set command through ALVIS? It turns out that this raises an interesting issue. Let us assume that the player presently with the football is "player2." We could double-click on "player2" in the *Storyboard* view, which would bring up an *Edit Cutout* dialog box that resembles the *Create Cutout* dialog box presented earlier. We could then select "red" from the "Outline Color" drop-down menu and dismiss the dialog box. ALVIS would then generate the following command: *set outlinecolor of player2 to red*. Notice that what ALVIS generates is more specific than what I have written above—an observation that underscores the fact that ALVIS is too naïve to generate general-purpose code from a specific instance.

```
 1: --create the storyboard elements
 2: create cutout player1 -graphicrep "cutout.cut" -data "300"
 3: create cutout player2 as clone of player1 -data "250"
 4: create cutout player3 as clone of player1 -data "200"
 5: create cutout player2 as clone of player1 -data "150"
 6: create cutout referee -graphicrep "referee.cut" -data "referee"
 7: create cutout football -graphicrep "football.cut"
 8: create cutout goal -graphicrep "goal.cut" -outlinecolor red
 8: create grid a -rows 1 columns 4 -cellwidth width of player1
 9: -cellheight height of player1
10: --place the storyboard elements
11: place a at 0.5,0.5
12: place player1 at position 1 of a
13: place player2 at position 2 of a
14: place player3 at position 3 of a
15: place player4 at position 4 of a
16: place referee at
17: place football at top-center of referee
18: --ready to perform bubble sort. . .
19: while outlinecolor of cutout at position 2 of a is not red --do outer loop
20:   move football to left-center of cutout at position 1 of a
21:   while outlinecolor of cutout right-of cutout touching football is red
22:     --do inner loop; swap or fumble
23:     if data of cutout touching football >
24:         data of cutout right-of cutout touching football
25:       --swap players
26:       assign p1 to position in a of cutout touching football
27:       assign p2 to position in a of cutout right-of cutout touching football
28:       doconcurrent
29:         move cutout touching football to p2
30:         move cutout right-of cutout touching football to p1
31:          move football right cellwidth of a
32:       enddoconcurrent
33:     else --fumble ball to next player in line
34:       move football right cellwidth of a
35:     endif
36:   endwhile  --inner loop
37:   --ball carrier has reached rightful place:
38:   set outlinecolor of cutout touching football to red
39:   --toss ball back to referee, in preparation for next pass of outer loop
40:   move football to top-center of referee
41: endwhile --outer loop
42: set outlinecolor of cutout at position 1 of a to red
```

Figure 53. SALSA Script for Football Bubble Sort Storyboard

## Presenting the Storyboard

Having programmed the football bubble sort storyboard, I now proceed to execute and present it. I begin by choosing "Reset at Beginning" on the *Present* menu, which resets the script to the beginning. Since nothing interesting occurs in the script until after the *Storyboard* view is populated, I elect to execute the script to the first line past the last "place" command (line 19 in Figure 53) by positioning the cursor on that line and choosing "Run to

Cursor" from the *Present* menu. The first time through the algorithm's outer loop, I want to walk through the script slowly, so that I can explain the inner logic carefully. Thus, I step forward slowly, pausing to explain the comparison (lines 23–24 in Figure 53) and subsequent exchange (lines 26–32 in Figure 53) of the first two players in line, and using the "Presentation Pointer" tool to point at the storyboard as I explain. Before the two players are swapped, I use the "Markup Pen" tool to circle the two elements that are about to be swapped; Figure 54 shows the ALVIS environment at this point.

Now suppose an audience member wonders whether it might be useful to flash the players as they are being compared. Thanks to ALVIS's flexible animation interface and dynamic modification abilities, I am able to test out this idea on the spot. As I attempt to edit line 22 of the script, ALVIS recognizes that a change at this point necessitates a re-parsing of the entire while loop. As a result, ALVIS uses a dialog box (Figure x) to inform me that in order to edit that line, I first have to back up the script to just before the while loop ALVIS. I click on "Yes," and ALVIS automatically backs up the script to the point just before the while loop (line 18 in Figure 53). I then add four lines of code beginning on line 22:

```
doconcurrent --flash players to be compared
        flash cutout touching football for 1 sec
        flash cutout right-of cutout touching football for 1 sec
enddoconcurrent
```

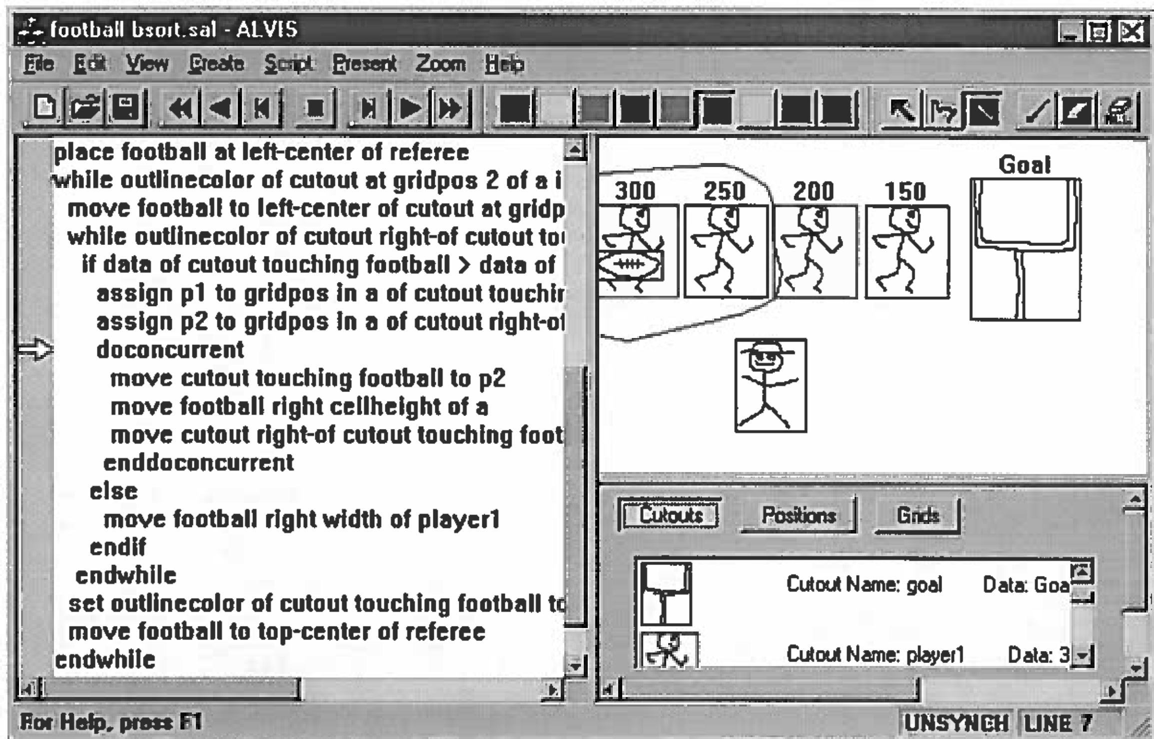



Figure 54. The ALVIS Environment Before the Players Are Swapped

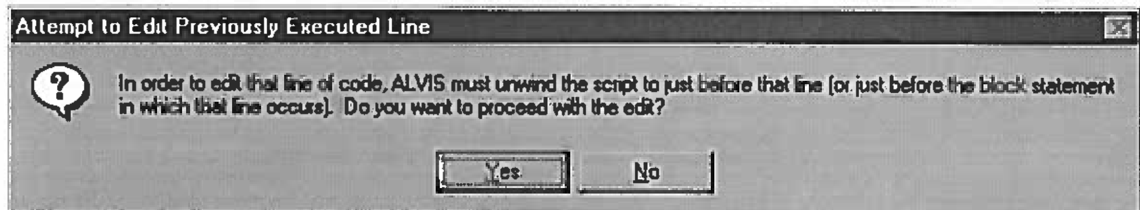



Figure 55. "Attempt to Edit Previously Executed Line" Dialog Box

I can now proceed with my presentation without having to recompile the script, and without even having to start the script over from the beginning.

## Related Work

SALSA and ALVIS have key features in common with a family of systems geared toward rapidly prototyping graphical user interfaces. QUICK (Douglas, Doerry, & Novick, 1992) supports a direct manipulation interface for programming graphical applications (which may include animation) in terms of an underlying interpreted language with explicit support for spatial relations. SILK (Landay & Myers, 1995) and PatchWork (van de Kant, Wilson, Bekker, Johnson & Johnson, 1998) are sketch-based systems that support the rapid construction of low-fidelity interface prototypes; the latter's notion of "patches" is similar to SALSA's concept of "cutouts."

Beginning with Brown's BALSA system (Brown, 1988), a legacy of interactive AV systems have been developed to help teach and learn algorithms (see, e.g., Helttula, Hyrskykari, & Raiha, 1989; Naps, 1990; Roman, Cox, Wilcox, & Plun, 1992; Stasko, 1989; Stasko, 1997). SALSA and ALVIS differ from these systems in three fundamental ways. First, whereas the design of past systems has been based mainly on designer intuitions (so-called "armchair design"),[58] I have made an earnest effort to get out of the design armchair—that is, to root the design of SALSA and ALVIS firmly in empirical data. Specifically, as part of a user-centered design process, I have used observations from several empirical studies as a basis for formulating the functional and usability requirements of the language and system. Moreover, the observations also clearly inspired ALVIS's conceptual model, which is rooted in "paper-and-pencil" storyboard construction. Finally, before I began

---

[58]A noteworthy exception to this is Mukerjea and Stasko's (1994) work on the Lens system. To assist them in designing Lens's animation language, they studied 42 actual visualizations developed by course instructors.

implementation, I verified and refined the syntax and semantics of SALSA through a pilot study involving students enrolled in CIS 315.

Second, the AV construction technique pioneered by ALVIS and SALSA differs from those supported by existing systems. To place ALVIS and SALSA into perspective, Figure 56 presents a taxonomy of AV construction techniques. These may be divided into three main categories—*algorithmic*, *spatial*, and *manual*—as described below.

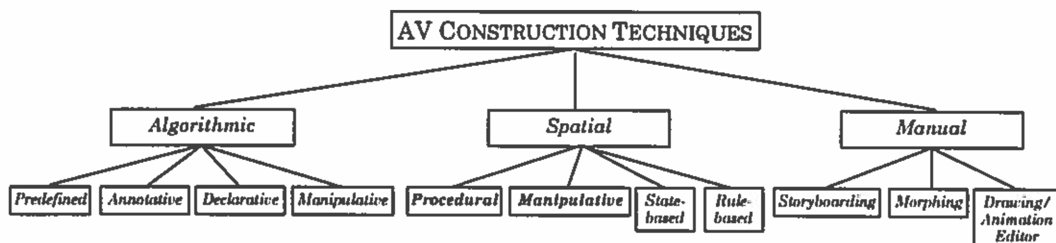



Figure 56. A Taxonomy of AV Construction Techniques

*Algorithmic.* Algorithmic AV construction involves the specification of mappings between an underlying (implemented) algorithm, and a visualization. In Chapter II, I called this technique *direct generation*, since it aims to generate a visualization directly as a byproduct of algorithm execution. Variations on algorithmic AV construction include (a) *predefined* techniques, in which one chooses when and what to view, but has no choice with respect to the form of the visualization (see, e.g., Myers, 1983; Naps, 1990); (b) *annotative* techniques, in which one annotates an algorithm with event markers that give rise to updates in the visualization (see, e.g., Duisburg, 1987a,b; Brown, 1988; Stasko, 1989; Stasko, 1997); (c) *declarative* techniques, in which one specifies a set of rules for mapping an executing program to a visual representation (Roman, Cox, Wilcox, & Plun, 1992); and (d) *manipulative* techniques, in which uses some form of direct manipulation and dialog box fill-

in to map an algorithm to a visual representation (see, e.g., Mukherjea & Stasko, 1994; Stasko, 1991).

*Spatial.* In stark contrast to algorithmic AV construction, spatial AV construction completely abandons the notion of an underlying "driver" algorithm that generates a visualization as a byproduct of its execution. Instead, spatial techniques aim to specify a visualization in terms of the *spatiality* of the visualization itself. By providing a language in which one can write a procedural specification of a visualization in terms of the visualization's spatiality, SALSA pioneers a *procedural* approach to spatial AV construction. As a graphical front-end to SALSA, ALVIS supports a *manipulative* approach that allows many components of SALSA programs to be programmed by direct-manipulation. Likewise, two lines of related research explore *manipulative* approaches to spatial AV construction; however, the underlying programming paradigm differs in both cases. First, Michail's Opsis (1996) supports the construction of binary tree algorithms through the manipulation of abstract visual representations of binary trees. In particular, Opsis users construct a binary tree program (i.e., a visualization) by specifying transformations between abstract program (i.e., visualization) states; this might be called a *state-based* approach. Second, Erwig (1991) and Brown and Vander Zanden (1998) describe systems that support a *rule-based* approach; users construct visualizations by specifying *rewrite rules*, which stipulate how a particular frame of the visualization should be transformed into a new frame.

*Manual.* In *manual* AV construction, one abandons a formal underlying execution model altogether. As a result, and in contrast to the *algorithmic* and *spatial* techniques, there exists no chance of constructing AVs the work for general input; AV execution is entirely under human control. *Storyboarding*, the manual technique on which ALVIS and SALSA are based, involves the use of simple art supplies to construct a visualization. In contrast, Citrin and Gurka (1996) describe a manual technique in which a drawing editor,

coupled with off-the-shelf morphing software, is used to produce animations for classroom demonstration. Finally, *drawing and animation editors* such as CorelDraw and Macromedia Director might be used to construct visualizations. For example, in my ethnographic studies (see Chapter IV), I observed several students who used drawing editors to construct their storyboards. In a similar vein, Brown and Vander Zanden (1998) describe a more specialized drawing editor, whose built-in semantics support the construction of data structure drawings that can be interactively manipulated to demonstrate common operations.

The third key difference between SALSA and ALVIS and existing AV technology lies in their support for AV presentation. Almost without exception, existing AV technology has adopted the standard animation control interface pioneered by Brown (1988). As discussed in Chapter II, that interface allows one to start and pause the animation, step through the animation, and adjust animation speed. ALVIS's animation control interface goes beyond this interface by supporting both forward and backward execution; this allows the user to back up and review segments of the animation in response to audience requests. In addition, with the notable exception of Brown and Vander Zanden's (1998) Whiteboard Environment, no existing AV system allows one to annotate an animation as it is running. Finally, no existing AV system supports the dynamic modification of animation; rather, modifying an animation most often entails changing and recompiling source code, which is seldom feasible within the scope of an interactive presentation.

## Summary

This chapter has presented SALSA and ALVIS, a prototype language and system that manifest the design implications of the set of hypotheses proposed in Chapter V. As I have illustrated, taking these hypotheses seriously leads to a design that differs, in three

fundamental ways, from that of extant AV technology. First, rather than supporting highly polished AVs that resemble textbook figures, the prototype supports rough sketches, which require far less overhead to create, and, according to my ethnographic fieldwork, stimulate more relevant conversations about algorithms. Second, the prototype pioneers a novel AV specification technique in which users construct AVs by making use of and explicitly testing the spatial relations among constituent objects. This technique more closely models the way in which humans have been observed to construct and execute storyboards (non-problematically) using simple art supplies. Finally, in departure from existing AV technology's exclusive focus on supporting AV creation, the prototype presented here focuses extensively on supporting the process of AV presentation. Taking AV presentation seriously leads to the three features not supported by existing technology: forward and backward execution, dynamic mark-up, and dynamic modification.

# CHAPTER VIII

# CONCLUSION

> We learn 10 percent of what we read, 20 percent of what we hear, 30 percent of what we see, 50 percent of what we both see and hear, 70 percent of what is discussed with others, 80 percent of what we experience personally, 95 percent of what we teach to someone else.
> W. Glasser [as quoted in (Griss, 1999)]

The advent of interactive, graphical workstations in the early 1980s spurred much initial excitement and enthusiasm among computer science educators. Finally, improved processing speeds and enhanced displays made it possible to produce, with relatively little effort, a clear illustration of how an algorithm works. Even more exciting, these illustrations could be produced automatically as a byproduct of an algorithm's execution, thereby guaranteeing their accuracy. Computer science educators thus had high hopes that, when used as a visual aid in lectures, or when given to students to use as a study aid, computer-generated algorithm animations not only would help students to learn algorithms better, but also would make instructors' jobs easier.

As I have illustrated, AV technology has failed to deliver on its initial promise of better learning and easier teaching. Research studies designed to substantiate the pedagogical benefits of AV have yielded mixed results, and AV technology has failed to see widespread use in computer science education. This has left AV technologists and computer science educators scratching their heads and asking, What went wrong? Why has AV technology failed to deliver?

If the wealth of published AV research in the decade after the introduction of the original interactive AV system (see Brown, 1988) is any indication, that question has been on

the mind of many researchers. A review of this work suggests that at least three alternative approaches to addressing the question have been considered:

1. Create better AV technology.
2. Create better pedagogical exercises involving AV technology.
3. Revamp the methods used to evaluate the effectiveness of AV technology.

The thesis for which I have argued here maintains that the theory of effectiveness guiding these three approaches is deficient. That theory of effectiveness, which I have labeled Epistemic Fidelity Theory, holds that the value of AV technology lies in its excellent ability to capture an expert's mental model of how an algorithm works, thus leading to the robust and efficient transfer of that mental model to students who view the AV. My argument has been that any approach that merely alters surface features of AV technology design, pedagogy, and evaluation, while remaining faithful to EF Theory as an underlying theoretical foundation, is bound to fail. Rather, if we are to make progress, we must first address the problem at its roots—by adopting a more appropriate guiding theory of effectiveness.

To explore this thesis, I have set about to motivate, construct, and refine an alternative theoretical foundation for AV technology, on which further research can build. At each decision point in this theory-building venture, my main objective has been to base my choices, to the greatest extent possible, on empirical findings, rather than on intuition. The first such decision came when I selected constructivism as a more appropriate guiding theory. Through a meta-study of past studies of AV effectiveness, I found that active learner involvement appeared to be the most significant factor in successful learning outcomes. Since constructivism furnishes a well-established, coherent account of why this might be, I opted to use it as my starting point. Moreover, observing that the target skills of an undergraduate algorithms course are inherently *social*, I chose to adopt the sociocultural

version of the theory, which views the locus of learning not at the level of the individual, but at the level of the community of practice, and which sees the value of AV technology not in its ability to transfer knowledge, but rather in its ability to facilitate access to more expert forms of participation in a community of practice.

The second decision came when I had to choose what to do with sociocultural constructivist theory. I decided that taking the theory seriously required a commitment to using ethnographic field techniques to carry out a naturalistic study of the community of practice being reproduced through an undergraduate algorithms course. Only through such a study would I be able to observe the effects of the sociocultural constructivist approach to using AV technology, which invites students to participate as (expert) instructors by using AV technology to construct and present their own AVs.

Traditional AV technology facilitates the construction of *high epistemic fidelity* AVs—that is, AVs that work for general input, and that have the polished look of textbook figures. My ethnographic studies found that traditional AV technology is ill-suited for this pedagogic approach, not only because the construction of high epistemic fidelity AVs requires inordinate amounts of time, but also because it requires students to engage in activities that are irrelevant to the focus of an undergraduate algorithms course. In contrast, when students used simple art supplies to construct and present their own, low epistemic fidelity AVs, they spent significantly less time overall, and the time that they did spend was dedicated to more relevant activities. An additional, unexpected benefit of low epistemic fidelity AVs was that they stimulated more relevant conversations about algorithms than did high epistemic fidelity AVs.

These findings led to a third decision: How to articulate the revised theoretical position? Remaining true to the findings of the ethnographic studies, I opted to express the alternative theoretical position in terms of four hypotheses that link cause to effect:

1. *The Activity Relevance Hypothesis*: Low input generality, low typeset fidelity, and direct graphics cause high activity relevance.
2. *The Communication Effectiveness Hypothesis*: Low epistemic fidelity causes high communication effectiveness.
3. *The Understanding and Recall Hypothesis*: Self-constructing AVs with a story line, and then presenting them to an instructor for feedback and discussion, causes high recall and understanding.
4. *The Community-Building Hypothesis*: Self-construction and high instructor communication cause high community-building.

These hypotheses are the centerpiece of the dissertation; they outline the provisional theoretical position that it proposes. However, far from being the final word on AV effectiveness, the hypotheses raise more questions than they answer. This observation brought me to a fourth decision: Which questions might I answer within the scope of this dissertation, and which ones would be best left to future research? In making this decision, I attempted to choose two research activities that would hold the most promise in demonstrating the plausibility and desirability of the theoretical position.

First, realizing that the community of computer science educators would probably be most receptive to empirical evaluations that consider traditional learning outcomes, I opted to conduct a controlled experiment in which I put to the test a key piece of the Understanding and Recall Hypothesis: On a test of procedural understanding and recall, students who construct their own AVs will outperform students who interact with an AV constructed by an expert. While the experiment failed to demonstrate a significant difference between the two treatment groups, the fact that the students who constructed their own AVs out of simple art supplies slightly outperformed the students who interacted with a predefined AV should be of interest to those instructors looking for a low-overhead

way of incorporating AV technology into their courses. Indeed, the result suggests that this simple approach, which requires no technology whatsoever, will help students learn just as well as the traditional, more costly approach of having students interact with computer-based AVs.

Second, recognizing that important implications of the hypotheses for technology design, I decided to build a prototype AV system rooted in the hypotheses. While the hypotheses have implications with respect to *what* tasks system users should be able to accomplish with the system (viz., the creation and presentation of low epistemic fidelity AVs), they offer little guidance with respect to *how* users should accomplish those tasks. Thus, I found myself at final decision point: How to design the system? Once again, empirical data guided my decision; I drew from the key findings of several empirical studies in order to develop the detailed design of the system. As we have seen, the result of the design exercise was SALSA, a high-level interpreted language for programming low epistemic fidelity AVs, and ALVIS, a graphical front end for programming SALSA scripts by direct manipulation.

Having summarized the trajectory this dissertation has taken, I now elaborate on its research contributions.

## Research Contributions

The primary research contribution of this dissertation is the novel theoretical position articulated by the framework of cause and effect presented in Chapter V. The Community-Building Hypothesis can be characterized as a specialized version of sociocultural constructivism that specifically addresses the pedagogical use of AV technology in an undergraduate algorithms course. Likewise, the Understanding and Recall Hypothesis

specializes sociocultural constructivism, which recognizes self-construction and presentation as crucial to learning insofar as they *vest* students in their own learning (Lave, 1997). In addition, the aspect of the Understanding and Recall Hypothesis that states that constructing an AV leads to high recall and understanding can be seen as a specialization of two other existing theories: (a) cognitive constructivism, which views learning as students' active construction of their own understandings, and (b) mnemonic theory, which underscores the value of stories in aiding memory. Finally, in contrast to the Community Building and Understanding and Recall Hypotheses, the Activity Relevance and Communication Effectiveness Hypotheses do not derive from past theoretical positions. Rather, they are quite specific to learning in an algorithms course, identifying the particular conditions under which AV construction and AV presentation will be most effective as learning activities within that context.

The framework hypotheses plainly have important implications for algorithms pedagogy. The second major contribution of this dissertation is a set of guidelines, embodied in the framework hypotheses, that outline a practical (low overhead, low cost) pedagogical approach to incorporating algorithm visualization into an algorithms course. Specifically, the pedagogical approach recommended by the hypotheses models the approach taken in Study II (see Chapter IV): Have groups of students use simple art supplies to construct AVs that illustrate a target algorithm for a few, carefully chosen input data sets, and then ask them to present their storyboards to the class for discussion and feedback.

In the process of arriving at the framework hypotheses, this dissertation makes two secondary research contributions. First, the meta-analysis of past empirical studies of AV effectiveness that I present in Chapter II constitutes the first attempt to synthesize and identify trends in the body of work as a whole. Second, the two ethnographic studies that I present in brief in Chapter IV, and in detail in Appendices A and B, are the first of their

kind. In particular, they are the first to consider the community aspects of AV technology, and they are the first to do so in a conscientious, systematic way—that is, by using a collection of established research techniques, and by taking care to explain precisely how the techniques were used.

Finally, the two research directions I pursued in response to the framework of cause and effect make two other secondary research contributions. First, the experiment presented in Chapter VI suggests that the self-construction factor, by itself, may not be strong enough to significantly impact procedural understanding and recall. Second, the system-building effort described in Chapter VII constitutes the first serious attempt to ground the design of an AV system firmly in empirical data. Moreover, the resulting prototype language and system, SALSA and ALVIS, demonstrate three design features that are absent in existing AV technology:

1. support for a novel way of specifying an AV in terms of spatial logic;
2. a novel control interface that supports both forward and backward execution; and
3. a novel presentation interface that supports dynamic mark-up and modification of an AV, even while it is executing.

As is the case for any substantial research project, the most significant contribution of this work may well be the wealth of research questions it raises for future work. I consider these in the following section.

## Directions for Future Research

The framework of cause and effect presented in Chapter V sketches out a provisional theoretical position that clearly requires further development and refinement. To that end,

the two research projects presented in Chapters VI and VII barely scratch the surface. Indeed, each framework hypothesis invites further exploration, as discussed below.

## The Understanding and Recall Hypothesis

Recall that the Understanding and Recall Hypothesis states that student self construction, instructor communication, and AV story content lead to improved procedural understanding and recall. In Chapter VI, I proposed a research program for systematically exploring the Understanding and Recall Hypothesis. Specifically, the research program included a series of three planned experiments, each of which manipulates one of the key causal factors of the Understanding and Recall Hypothesis while holding the others constant. The experiment presented in Chapter VI, which examined the impact of student self-construction, constitutes the initial step in this research program. The other two proposed experiments, which would examine the impact of instructor interaction and story content, remain as key pieces of future research.

In addition, the main result of the experiment of Chapter VI—that self-construction, by itself, does not appear strong enough to effect improved procedural understanding and recall—suggests that it may be important to explore interaction effects in future experiments. For example, one promising 2 × 2 design (see Table 14) would simultaneously explore the effects of the self-construction and instructor communication factors. This experiment would include four treatment groups: the two included in the experiment presented in Chapter VI (Self Construction, Active Viewing), as well as two others in which students also had the opportunity to interact with an instructor (Self Construction + Instructor Communication, Active Viewing + Instructor Communication). An analysis of variance could be performed to determine not only if significant differences exist among

treatment groups, but also whether there exist interaction effects. The Understanding and Recall Hypothesis would predict an interaction effect between the Self Construction and Instructor Communication factors: Students who construct their own AVs and then discuss them with an instructor will significantly outperform students who interact with a predefined AV and do not have an opportunity to interact with an instructor.

Table 14. A 2 × 2 Design that Explores Self Construction and Instructor Communication

| | | Self-Construction | |
|---|---|---|---|
| | | No | Yes |
| Instructor Communication | No | 12 | 12 |
| | Yes | 12 | 12 |

The Activity Relevance Hypothesis

Recall that the Activity Relevance Hypothesis states that students who build low input generality, low typeset fidelity AVs using direct graphics engage in more relevant activities than do students who use quantitative graphics to produce input-general, high typeset fidelity AVs. This hypothesis was informally verified in the ethnographic fieldwork presented in Chapter IV. Specifically, through observation, interviews, and diary analysis, I found not only that the storyboard construction exercises required far less time than did the Samba construction exercises, but that they 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 systematic, controlled investigation. For example, one could conduct an experimental study in which a sample of algorithms students is divided into two groups: Samba and Storyboard. Pairs of

students in the Samba group are required to construct an input general AV in Samba, while pairs of students in the Storyboard group are required to construct an art supply storyboard that works for just a few, carefully chosen input data sets. So that all construction activities could be videotaped for later analysis, students would be required to work on their AVs only in a closed laboratory during scheduled sessions. Post-hoc Interaction Analysis (Jordan, 1995) of the videotapes could be used to develop a classification scheme for student activities, and to determine precisely how much time each pair of students spent doing each activity type. In addition, the relative relevance of each activity type could be determined through consultation with one or more algorithms instructors. This would allow one to conduct statistical tests to determine whether significant differences exist between the two groups with respect to the relevance of the activities in which they engaged.

## The Communication Effectiveness Hypothesis

Recall that the Communication Effectiveness Hypothesis states that low epistemic fidelity AVs stimulate more relevant conversations about algorithms than do high epistemic fidelity AVs. In the ethnographic studies presented in Chapter IV, I informally verified this hypothesis through a post-hoc analysis of videotaped footage of the student presentation sessions.

As is the case for Activity Relevance Hypothesis, this hypothesis could clearly be verified more rigorously through a systematic investigation. In fact, one could do so by extending the study of activity relevance just outlined. In particular, after they create their AVs, each pair of students could be required to present their AVs to an audience that includes the entire group of students, along with a course instructor. Once again, all presentation sessions would be videotaped in order to facilitate post-hoc analysis, as follows.

First, through post hoc Interaction Analysis (Jordan, 1995), one could develop a scheme for classifying the topical content of the monologues and conversational exchanges that take place in the presentations; the relative relevance of each topic could be determined in consultation with one or more algorithms instructors. The scheme could then be used as a basis for precisely determining the proportion of time in each presentation that is dedicated to each topic. Ultimately, statistical tests could be used to determine whether significant differences in topic relevance exist between presentations of low and high epistemic fidelity AVs.

Recall that the second measure of communication effectiveness that I proposed in Chapter V is *mutual intelligibility*—the extent to which an AV serves as a resource that assists conversational participants in establishing a shared understanding of the algorithms being presented. Here, a modified form of *breakdown analysis* (Doerry, 1995) could be used to determine the amount of communicative breakdown (i.e., points where mutual intelligibility is lost) in each presentation; a statistical test could be used to detect a difference between levels of mutual intelligibility in presentations of low and high epistemic fidelity presentations.

## The Community-Building Hypothesis

Recall that the Community-Building Hypothesis causally links self-construction and instructor communication to community-building. Drawing from sociocultural constructivism, the notion of community-building used here refers to the reproduction of the Community of Schooled Algorithmaticians (COSA) through an undergraduate algorithms course. Through that process of reproduction, community newcomers—the students enrolled in the course—gradually move toward full community membership by participating

in increasingly expert ways in community activities—namely, by constructing and presenting their own AVs.

While it is unlikely that any one student could gain full membership in the COSA (that is, a level of membership comparable to that of a course instructor) by the end of a single algorithms course, it should certainly be possible to observe tangible differences in a student's level of membership and identity over the course of the academic term. Therein lies the challenge in systematically validating this hypothesis: How does one measure such nebulous concepts as "level of community membership" and "identity?" 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* to estimate the extent to which students' identity is wrapped up in the COSA. Appendix G presents an example of a questionnaire that might be used to gauge attitudes toward an undergraduate algorithms course; I developed it by adapting the "Revised Math Attitude Scale" developed by Aiken and Dreger (1961), and reprinted in (Shaw & Wright, 1967). The methodology for using such a questionnaire would be to give it to two student groups, each of which is enrolled in an alternative offering of an undergraduate algorithms course. In one of the course offerings, students would use AV technology in the traditional way—by interacting with predefined AVs. In the other of the course offerings, students would use AV technology in the way argued for in this dissertation—by constructing and presenting their own AVs. The questionnaire would be administered to both groups twice: once at the beginning of the academic term, in order to assess baseline attitudes, and again at the end of the academic term, in order to gauge students' attitudes at the end of the course. The results of the exit questionnaire could then be compared against the results of the baseline questionnaire in order to compute the relative change in attitudes for each group. Finally, a

statistical test could be used to determine whether a significant difference exists between the two groups' attitudinal changes. A statistically significant difference would be grounds for concluding that one pedagogical treatment better facilitates community-building than the other.

Second, and more ambitious, future research might explore the possibility of using Cultural Consensus Theory (Romney, Weller, & Batchelder, 1986) as a means of calculating one's level of community membership. Cultural Consensus Theory recasts one's level of community membership in terms of one's level of *agreement* with community members on matters of importance to the community. To calculate agreement, the theory derives a formal statistical model for assessing the "cultural competence" of the individual members of a community of practice. The basis for such an assessment are the answers that a sample of community informants furnish to a set of questions. The questions must be carefully chosen so that they address a body of knowledge on which the community of practice is assumed to agree. However, unlike the statistical models traditionally applied to test-taking, Consensus Theory's statistical model does not assume an objective truth against which informants' answers are to be measured. Rather, the model uses informants' answers as a basis for constructing a *cultural truth*, according to which informants' cultural competence can then be assessed.[59]

Consensus Theory's statistical model constructs such a cultural truth based on patterns of agreement among informants. The assumption is that "the correspondence between the answers of any two informants is a function of the extent to which each is correlated with the [cultural] truth" (Romney, Weller, & Batchelder, 1986, p. 316). In other words, the most central members of the community, whose cultural knowledge is the most

"complete," are highly likely to agree with each other by offering identical answers to the questions. On the other hand, less central members of the community are less likely to agree both with each other, and with central members of the community.

To gauge the ability of AV-based pedagogical exercises to facilitate community-building, one can employ a strategy similar to the strategy I outlined for using the attitude questionnaire. First, divide a sample of computer science students into two groups, each of which enrolls in an alternative offering of an algorithms course. One of the offerings makes use of AV technology in the traditional way, and the other offering makes use of AV technology in the sociocultural constructivist way explored in this dissertation. Next, measure each group's cultural competence twice—once at the beginning of the course to assess their baseline competence, and once at the end of the course to assess their final competence. Finally, statistically compare the two groups' mean change in competence. If a significant difference exists, it would be grounds for concluding that one of the pedagogical treatments better facilitates community-building than the other.

Of course, if Cultural Consensus theory is to be used as the foundation for such a study, one must first design an appropriate instrument for assessing COSA cultural competence. In this dissertation, I have identified three key forms of COSA participation that are mediated by AV technology:

1. *AV reading*—meaningfully interpreting a graphical representation of an algorithm;
2. *AV writing*—constructing one's own graphical representation of an algorithm; and

---

[59]Hence, the use of the term *informant* (as opposed to, say, *subject*) is deliberate; it underscores the fact that participants in a Consensus Study are *informing* the researcher of their culture, rather than the researcher *subjecting* them to a test.

3. *AV presentation/discussion*—presenting an AV to community members for discussion.

Clearly, if the COSA really is a distinct community of practice, one would expect a high level of agreement among its members with respect to the way in which they perform these tasks. For example, given an AV, one would expect members of the community to "read" the AV in a similar way.[60] Likewise, given an algorithm, one would expect COSA members to "write" similar AVs that describe it—that is, AVs that capture the visualization in terms of a similar semantics.[61] The challenge, of course, lies in coming up with a way to characterize agreement in these tasks *quantitatively*, so that results can be fed to Consensus Theory's statistical model.

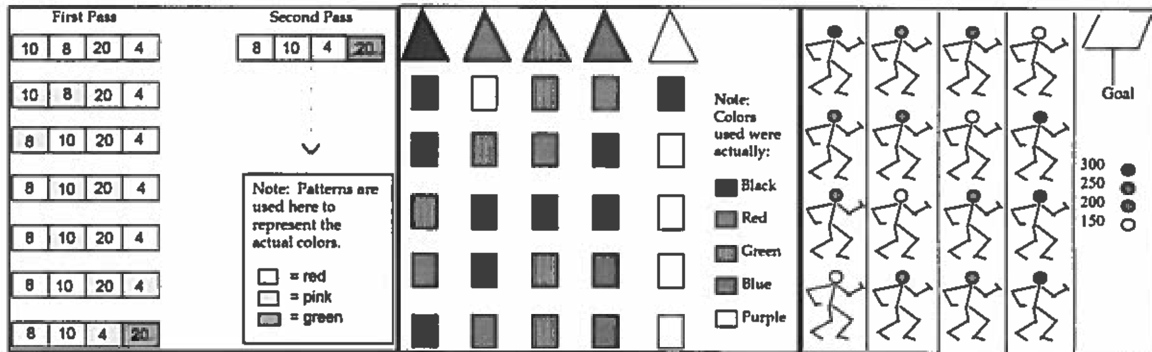
I propose the following as a technique for quantitatively assessing the level of agreement among COSA members' AV reading and writing activities. The technique draws on the *semantic level analysis* technique first introduced by Douglas, Hundhausen, and McKeown (1995, 1996). Given a set of AVs that describe an identical algorithm, semantic level analysis involves mapping the lexical entities, attributes, and transformations of the AVs to the underlying semantics of the algorithm. Algorithm semantics is expressed in terms of a pseudocode-like description of the algorithm; each mapping between a lexical item of an AV and a variable or statement in the underlying pseudocode-like description is called a *semantic primitive*.

Figure 57 presents an example of the technique taken from the Douglas, Hundhausen, and McKeown studies. In this example, three alternative AVs of the

---

[60]For empirical evidence that central members of a community of practice read the representations of the community in a similar way, see (Petre & Green 1993), who studied digital circuit designers.

[61]This is, in fact, borne out by recent empirical studies that explored the human visualization of algorithms; see (Chaabouni, 1996; Douglas, Hundhausen, & McKeown, 1995, 1996).




(a) Snapshots of three AVs of the bubblesort algorithm

| SEMANTICS | NUMBER AV LEXICON | COLOR AV LEXICON | FOOTBALL AV LEXICON |
|---|---|---|---|
| Sorting element | Square | Square | Stick Figure |
| Magnitude of element | Number symbol | Color | Color (as weight) |
| Array of elements | Contiguous row of squares | Non-contiguous row of squares | Contiguous row of figures |
| Inner loop pass history | Rows of sorting elements | — | — |
| Outer loop pass history | Columns of rows | Rows of sorting elements | Rows of sorting elements |
| Legend explicating ordering on sort elements | — | Triangles with color spectrum | Column of color/player weight pairs |

(b) Mappings between lexical entities and attributes of the bubblesort AVs and their semantics

| SEMANTICS | NUMBER AV LEXICON | COLOR AV LEXICON | FOOTBALL AV LEXICON |
|---|---|---|---|
| DO outer loop | Start new column of rows | Create new row of squares | Create new row of football players |
| DO inner loop | Create new row of squares | — | — |
| a) Reference elements to be compared | Color elements pink | — | Location of football |
| b) Compare elements (same, <, >) | — | — | Intuitions about how player size relates to running, tackling, and fumbling |
| c) Exchange elements | Exchange numbers | Exchange colors | Ball carrier advances by tackling next football player in line (thereby exchanging positions with that player) |
| d) Don't exchange elements | — | — | Fumble football to next player in line |
| Terminate outer loop | Color square in correct order green | — | — |
| Terminate Sorting | Ordering of natural numbers, all squares green | Color squares match legend | Players ordered by weight |

(c) Mappings between lexical transformations of the bubblesort AVs and their semantics

Figure 57. Example of the Semantic-Level Analysis Technique

bubblesort algorithm (see Figure 57(a)] observed in the study are semantically analyzed. Figure 57(b) maps the lexical entities and attributes of each AV, while Figure 57(c) maps their lexical transformations. As the example illustrates, while all three AVs differ greatly at a lexical level, they all capture the bubble sort algorithm in terms of a similar set of semantic primitives.

Notice that this semantic level analysis technique makes it possible to quantify the level of agreement between two people on AV reading and writing tasks. With respect to AV reading, two people are said to agree to the extent that they can view an AV and perform a similar semantic-level analysis on the AV. More formally, let $s_1$ be the set of semantic primitives (i.e., lexical-to-semantic mappings) that informant $i_1$ gleans from viewing an AV, and let $s_2$ be the set of semantic primitives that informant $i_2$ gleans from viewing the same AV. Then the *proportion of agreement* between informants $i_1$ and $i_2$ can be defined as the proportion of semantic primitives in $s_1$ and $s_2$ that are identical:

$$\textit{Proportion of agreement}(i_1,i_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}$$

With respect to AV writing, agreement between two informants can be calculated in an analogous fashion: Two people are said to agree to the extent that the AVs that they construct have semantic primitives in common. More formally, let $s_1$ be the set of semantic primitives determined to exist in the AV of informant $i_1$ for algorithm $a$, and let $s_s$ be the set of semantic primitives determined to exist in the AV constructed by informant $i_2$ for algorithm $a$. Then the above equation expresses the proportion of agreement between the AVs of $i_1$ and $i_2$.

Clearly, additional research is needed in order to determine whether this approach to quantifying agreement on AV reading and writing tasks will ultimately prove workable. For example, a pilot study I recently conducted suggests that, while the semantic level analysis

technique is learnable, people engaged in AV reading tasks tend to use diverse languages to describe the semantic mappings in a given AV. The diversity of their descriptions makes it difficult to determine the extent to which two semantic level analyses are actually in agreement. One possible way to ensure uniformity among two people's semantic level analyses is to require them to work with identical descriptions of both the underlying algorithm, and the AV being read. The former might be in high-level pseudocode, while the latter might be in a high-level AV description language like SALSA.

## The Prototype Language and System

SALSA and ALVIS, the prototype language and system derived from the framework hypotheses (see Chapter VII), clearly constitute work in progress. Much future work remains to be done in order both to refine the design features that they demonstrate, and to explore their benefits in the real world. In particular, future work should focus its efforts in three key areas. First, SALSA and ALVIS are presently implemented just fully enough to illustrate the major design implications of the hypotheses, and to demonstrate their feasibility in practice. In their present state, they fall well short of fully implementing all of the features included in the original conceptual design specification. Some key features that remain to be implemented include (a) the scissors tool, the algorithms for which prove particularly challenging; and (b) the graph and binary-tree s-structs, which would provide users with more options for laying out cutouts.

Second, future research should subject the SALSA and ALVIS prototype to iterative usability testing in order to improve the usability of its design. Ideally, the prototype would be subjected to two or more rounds of usability testing; each round would ideally include three to five target users (undergraduate algorithms students and instructors). Third, it is

important to note that ALVIS was originally designed to be used with a *stylus* in order to enable users to manipulate cutouts in a way that more closely models that way in which they would do so in the real world. However, the prototype implementation is not presently set up to work with a stylus. Thus, future research would do well to incorporate a stylus into the prototype, and to explore its potential as ALVIS's input device.

Finally, and perhaps most importantly, SALSA and ALVIS, in their present states, are frail research prototypes that were constructed for the purpose of making a specific point; they were not implemented to be robust systems suitable for use in the real world. Thus, an important direction for future research is to improve the robustness of SALSA and ALVIS, so that they advance beyond frail prototypes, and toward finished systems that can be used in an actual algorithms course. As robust systems, SALSA and ALVIS can serve as the technological foundation for the algorithms class of the future that this dissertation envisions, as explained in the following section.

## A Vision for the Future: The "Algorithms Studio"

Taken as a whole, the research presented in this dissertation points toward an approach to teaching algorithms that differs markedly from the conventional, lecture-based approach that presently predominates on university campuses. Inspired by the research presented in this dissertation, I would like to close by sharing my vision for the future algorithms course.

My vision draws from the instructional model presently used to teach architectural design, which constitutes a prime example of the learning-by-participation model advocated by sociocultural constructivism. Unlike algorithms students, who spend most of their time attending lectures and studying on their own, architectural students spend most of their

time with their peers in an architectural studio, where they work on collaborative, directed design projects. Gathered around drafting tables, student groups work out their ideas using drafting paper, pencils, and cardboard models; other students are always around to answer questions, to bounce ideas off, or to discuss design alternatives. In addition, during scheduled review sessions, students present their work-in-progress to their instructor for feedback. During final review sessions, students present their final designs to their instructor and peers.

So, too, could it be in an algorithms course. While lectures would likely remain a necessary part of the curriculum, the length of the frequency of the lectures could be reduced in order to give students more time in the "algorithms studio." There, they would engage in collaborative algorithm design and analysis projects at their "drafting tables"—computer workstations with large displays appropriate for group collaboration. Using a conceptual algorithm design tool like SALSA and ALVIS, students could explore alternative ways of solving assigned algorithm design problems. As they worked in the studio, peers would always be around to bounce an idea off, answer questions, or discuss possible design strategies. Moreover, students could use ALVIS to present tentative or partial solutions to their peers and instructor—both informally during studio work sessions, and more formally during scheduled review sessions. Finally, in the spirit of the animation presentation sessions that were part of the algorithms courses I studied in my ethnographic fieldwork, students could use a projection device in the studio to present their final algorithm designs to their instructor and peers on a large screen. These review sessions would not only provide instructors with an important basis for assessing students' progress, but they would also, as demonstrated in my ethnographic fieldwork, set up ideal conditions under which to discuss the conceptual foundations of algorithms.

This alternative, studio-based model of algorithms learning could go a long way not only toward getting students more actively involved in their own learning, but also toward promoting the kinds of activities and communication that prove so vital to fostering students' enculturation into the Community of Schooled Algorithmaticans. Indeed, if there is a key idea to take away from this dissertation, it is about the power of using AV technology in a radically different way in algorithms education: not as a means of producing informative displays that accurately convey how algorithms work, but rather as a means of granting students' *access* to the central practices of a community concerned with the conceptual foundations of algorithms.

# APPENDIX A

## ETHNOGRAPHIC STUDY I

> We always have believed that having the students become more fundamentally involved with the animations would be beneficial. . . .What if students built the animations of algorithms themselves, as opposed to simply interacting with animations already prepared for them?
>
> (Stasko, 1997, p. 25)

The original purpose of the ethnographic studies summarized in Chapter IV was to explore two important research questions that past experience reports on student-constructed AV assignments (most notably, Stasko, 1997) fail to address:

1. What are the costs and benefits of assignments in which students construct their own AVs?
2. How might sociocultural constructivist theory be tailored so as to account for the benefits of such assignments?

To address these questions, I considered two separate offerings of a third-year undergraduate algorithms course in which animation assignments were part of the curriculum. 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, and which I preent in this appendix, 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, and which I consider in Appendix B.

Observations made during Study I provide several crucial insights into the costs and benefits of such assignments from the perspective of both students and the instructor. The most significant of these is that animation assignments, in the format proposed by past instructors, are actually a formidable *distraction* with respect to the objectives of an undergraduate course that emphasizes the theoretical and conceptual foundations of algorithms. In addition, resonating with socioconstructivism, Study I provides a preliminary account of why AV assignments might be beneficial. Contrary to EF Theory, Study I suggests that their value rests in the kinds of thinking, decision-making, and activities they promote—namely, those of an *algorithms teacher*.

## Background

CIS 315, which is entitled simply "Algorithms," typifies the 300-level algorithms course taught within the computer science departments of most 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 technique accordingly to solve the problem. Furthermore, the course 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.

As indicated by the sample syllabus included in Appendix A, the CIS 315 course in which I conducted my fieldwork revolved around three fifty-minute lectures per week. An additional 50-minute discussion period provided an opportunity for students, the teaching assistant (TA), 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 (or final programming project), and various algorithm animation assignments. The algorithm animation assignments differed substantially in each of the course offerings I observed; thus, I will discuss their specifics in a separate section associated with each term's fieldwork.

In the subsections that follow, I provide essential background on Study I; most of this background also applies to Study II, and hence will not be repeated in Appendix C. First, I describe how I arranged the fieldwork. Second, I discuss a couple of important disclaimers. Third, I provide essential background on the Samba algorithm animation language used for the assignments. Fourth, I cover my field techniques in detail. Finally, I briefly describe my informants.

### Getting in the door

A serendipitous event brought about the opportunity for my fieldwork. In the spring of 1996, John Stasko, a pioneer of algorithm animation and leading advocate for its educational use, came to Oregon State University to give a colloquium talk, which detailed his use of Samba for student-constructed animation assignments (see Stasko, 1997). Stasko's talk piqued Professor John Lane's[62] interest. Several months before the start of Lane's 300-level algorithms course the following winter, I approached John and offered him my services as an algorithm animation specialist. He agreed to add animation assignments to his 315 syllabus, and to model those assignments after the animation assignments described in Stasko's talk. In return, I agreed to install the Samba software, to help him write the specific algorithm animation assignment descriptions, and to provide advice on how to grade the assignments. My fieldwork in CIS 315 thus came to be.

### Disclaimers

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 the benefits and costs of AV assignments. The disadvantage is that the observations and findings do not necessarily apply to other algorithms courses taught by other instructors, taken by other students, and offered at other universities. In this chapter and the next, it is important to recognize this limitation of my observations and findings; plainly, great care must be taken in any attempt to generalize them beyond the particular algorithms course I studied.

Aside from having limited generalizability, this study was limited by the biases I brought to the field. As a computer science graduate student and graduate teaching assistant, I obviously shared greatly in the culture of my informants. Prior to my fieldwork, I had

---

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

already taken a course that was similar in flavor to CIS 315, and I had even taken another, related course (Automata Theory) with Professor Lane.

It is difficult to say just how this experience in computer science education biased my observations. At the risk of stating the obvious, I can say that I probably took a lot more for granted than I should have. Much of what I saw—the day-to-day lectures, the instructor-student interaction, and student programming in the lab—was old hat to me. Although I have always felt quite comfortable with the theoretical side of computer science (efficiency analysis and formal proofs), I can say that, at the time of my fieldwork, much of what I had learned about the foundations of algorithms was far from fresh in my mind. I had taken my undergraduate algorithms course some 9 years prior to this fieldwork, and my graduate algorithms course some two years prior to this fieldwork. Although, at some level, they were familiar to me, I found the particular algorithms, problem-solving techniques and proofs presented in CIS 315 to be challenging nonetheless. In fact, I remember getting noticeably sidetracked from my ethnographic objectives during some of the lectures, as I tried to anticipate a proof technique, or to grasp the efficiency analysis of a particular algorithm discussed in class.

In addition, I approached the fieldwork not only as an avid AV researcher, but as a former AV technologist who once had a great stake in a particular educational AV system. However, since my participation in the development of the GAIGS AV system at Lawrence University (see Naps & Hundhausen, 1991), I had grown increasingly cynical of the value of AV technology. At the time of my fieldwork, then, it is fair to say that I had a healthy degree of skepticism regarding the putative benefits of the technology.

More importantly, I had no stake in the success of the particular AV technology (*Samba*; see the following subsection) that I was studying in this fieldwork. In this sense, the bias I brought to this study was radically different from the biases brought to bear on previous experience reports of AV technology (which were invariably written by the same people who built the technology): I entered the fieldwork not as an *advocate* for the technology, but rather as an *observer* whose bias tended to be more that of a critic than that of an advocate.

### Using Samba for Algorithm Animation

Students used *Samba* (Stasko, 1997) to construct their animations for the animation assignments of Study I and Study II. A front-end interpreter to the Polka animation system (Stasko & Kraemer, 1993), Samba supports multiple-window, color, two-and-a-half dimension animations. The Samba interpreter takes as input a text file containing a series of Samba commands (one per line), and generates a corresponding animation. Various Samba commands create new objects (e.g., rectangles, circles, text), set the attributes of those objects (e.g., color, size, visibility), and animate objects smoothly from one position to another. In addition, Samba provides an easy means of defining concurrent animation; those animation commands that should be executed simultaneously are simply placed between curly brackets {}.

Typically, one animates an algorithm with Samba by first implementing the algorithm in a programming language of one's choice. Then, following the interesting events paradigm established by Brown (1988), one annotates the algorithm with statements that print out appropriate Samba commands at interesting points in the algorithm. By executing the algorithm, one thus generates a Samba trace file that, when fed to the Samba interpreter, produces an animation of the algorithm. Nearly all the animations in Study I, and a handful

of the animations in Study II, were produced using this *direct generation* (see Chapter II) technique.

Note that it is also possible to create algorithm animations in Samba without relying on an underlying implementation of the algorithm to be animated. One such technique is to write a "driver" program that generates the desired algorithm animation. Such a driver program is not the algorithm itself; rather, it is a program specifically tailored to producing an animation of the algorithm for a particular, pre-defined set of input data. A second way to create a Samba animation without an underlying algorithm is "by hand"—that is, by manually creating a text file of Samba commands. Students made extensive use of both of these alternative techniques in Study II.

## Field Techniques

Figure 58 depicts the physical area covered by my fieldwork, in which 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 Lab.

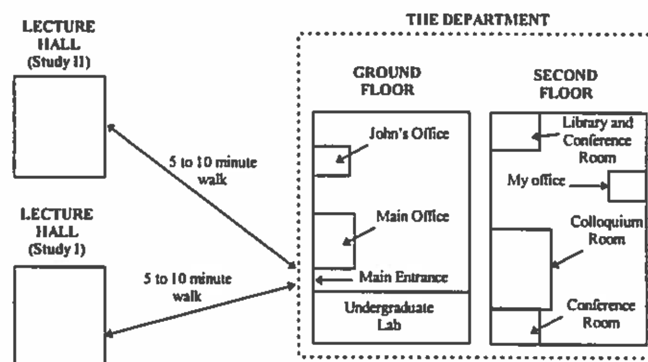



Figure 58. A Schematic of the Area in Which the Fieldwork Was Conducted

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 my dual-role of student observer and teaching assistant, I made use of a variety of ethnographic field techniques. Table 15 details the extent to which I used each of the techniques included in Wolcott's (1992) taxonomy of the three ethnographic "E's"—*experiencing*, *enquiring*, and *examining*. Below, I elaborate on the ways in which I made use of these techniques.

Table 15. Ethnographic Field Techniques vis-à-vis Wolcott's Taxonomy.

| "E" | Technique | Extent of use |
| --- | --- | --- |
| Experiencing | Active Participant | ■ |
| | Active Observer | ■ |
| | Passive Observer | ■ |
| | Brief or Spot Observation | □ |
| Enquiring | Casual | ■ |
| | Life History | □ |
| | Semi-structured Interview | ■ |
| | Survey | ■ |
| | Household Census | □ |
| | Projective Techniques | □ |
| | Other Measurement Techniques | □ |
| Examining | Using Archives | □ |
| | Using Other Written Documents | ■ |
| | Using Non-Written Materials | □ |
| | Collecting Art + Artifacts | ■ |
| | Keeping Fieldnotes, Journals | ■ |
| | Making Maps, Sketches | ▒ |
| | Recording on Videotape | ■ |
| | Recording on Audiotape | ■ |
| | Preparing a Field Draft | □ |

| Legend | | | |
| --- | --- | --- | --- |
| ■ Central to study | ■ Used some | ▒ Used a little | □ Not used or applicable |

## "Experiencing" Techniques

As a *student observer*, I employed active participation and active observation as primary field techniques. During Study I, I attended lectures as an active student participant. I vigorously took notes on the lectures, and even occasionally raised my hand and asked questions or provided answers. To my surprise, I seemed to blend in well in the lectures. In fact, during my first term of fieldwork, one student, who had recently learned of my expertise in algorithm animation, accosted me to ask whether I was actually enrolled in the course.

Through a voluntary e-mail list that I established during both quarters of my fieldwork, I elicited students interested in letting me observe them as they worked on their animation assignments. During Study I, I tracked one group of four students, along with two individual students. During the second quarter of fieldwork, I followed six student groups, each consisting of two to three students. Via e-mail that they carbon-copied to me, these students would agree on when and where to meet to work on their assignments; on occasion, I would show up and observe them. In all of these observation sessions, which took place in the Undergraduate Lab, I played the role of an active observer. In particular, I asked students questions about their animation projects as they worked on them, attempting to gain a sense

of the problems they were encountering, the issues they were facing, and their impressions of what they were doing.

As the *teaching assistant for algorithm animation*, I used active participation as a principal field technique. Prior to the start of the first academic term of fieldwork, I met with Professor Lane in his office on several occasions. In those meetings, we hashed out the details of the algorithm animation assignments, drawing both from his experience in teaching the class, and my experience with Samba and algorithm animation. After many of the lectures, I accompanied Professor Lane on the five to ten-minute walk back to the department from the lecture hall. During these back-to-the-department walks, we discussed a variety of issues of the day, including the upcoming schedule of algorithm animation assignment events (e.g., deadlines, grading sessions); questions regarding Samba; questions and concerns regarding how to grade animation assignments; and the potential for algorithm animation to illuminate the lecture topic of the day.

It is important to note that I never directly participated in the grading of student assignments. Such participation, in my view, would have posed a conflict of interest with my role as student observer. Although I was prepared to offer advice on grading, I seldom did; instead, Professor Lane tended to find his own answers to grading questions just by discussing his concerns with me.

Finally, during each term of fieldwork, I agreed to give an introductory lecture on algorithm animation in Samba. These lectures took place during the course's regularly-scheduled lab period. As the local expert on algorithm animation, I tried to make myself available to students who had questions or concerns regarding the animation assignments. In particular, I was "virtually" available via e-mail, and "physically" available after lectures, and by appointment in my office. Most students asked me questions via e-mail and after lectures, although a few did arrange to see me in my office.

"Enquiring" Techniques

In my interaction with both students and Professor Lane, I tended to ask a lot of questions on an informal basis. As important themes and issues emerged from those informal discussions, I would write them down for more structured inquiry at a later time. Twice during each of the academic terms of my fieldwork, I audiotaped semi-structured interviews with John. For these interviews, I prepared a list of themes I wanted to pursue; however, I also allowed the direction of our conversations to guide my inquiry. In addition, I conducted semi-structured interviews with three student informants and four informant groups. In most cases, these interviews followed participant observation sessions in which I gained a feel for their animation projects.

As I mentioned above, I established a volunteer "algorithm animation" e-mail list during each term of fieldwork. During both terms of fieldwork, I used this e-mail list to elicit students who would be willing to let me observe their algorithm animation activities. During the first term of fieldwork, I also used this mailing list to administer two brief on-line surveys (see Appendix A). These surveys elicited students' general impressions regarding the algorithm animation assignment, what activities they performed, and estimates of the amount of time spent on each activity. Nine students completed to the first survey, while three students completed the second survey.

"Examining" Techniques

Arguably, my fieldwork relied more heavily on "Examining" techniques than on either of the other techniques. I used four different "Examining" techniques. First, as described above, I audiotaped several semi-structured interviews during the course of the fieldwork. I subsequently transcribed all of these, from which I excerpt extensively in both this chapter, and in Chapter 4.

Second, I took extensive fieldnotes during both terms of the fieldwork. During the first term of fieldwork, many of these notes were written during Professor Lane's lectures. Interestingly, the content of my lecture notes evolved considerably over the course of the term. Whereas they probably resembled any other student's notes at the beginning of the term, they gradually took on the character of an ethnographer's fieldnotes as the term progressed. By the end of the term, my lecture notes did not capture the content of the lecture at all; rather, they included evolving ideas, theories, and questions that I wanted to pursue in subsequent fieldwork. In addition, Professor Lane would often talk a lot when I least expected it, and I often regretted that I did not have my tape recorder along with me more often. After several conversations with him, I remember rushing back to my office and frantically writing down what he had said, along with the flurry of ideas I had in response to what he said.

Third, during both terms of fieldwork, students were required to present their final animations to their instructor and fellow students. Several of these presentations turned out to stimulate lively dialogs, generating numerous comments and questions from John and the rest of the audience. I attended and audio- or videotaped all of these presentations, each of which lasted between 10 to 20 minutes. In addition, as I shall discuss in greater detail below, the animation assignment given during the second term of fieldwork also required students to present an "animation storyboard"—which might be composed of sketches, transparencies, or construction paper cut-outs—detailing their proposed animations. Students arranged to present their storyboards to Professor Lane and other interested students during sessions scheduled outside of the regular lectures. I both observed and videotaped all of these sessions.

Fourth, students deposited their animation assignments into a course directory that I set up and maintained. I was thus able to collect all student animations with minimal effort. Finally, for the Study II, students were required to hand in to me a diary documenting what they did for the animation assignment, what problems they encountered, and how much time they spent. I shall discuss the diaries in greater detail in Chapter 4.

## Informants: Students and the Instructor

Between 40 and 50 computer science majors were enrolled in each of the two CIS 315 classes I observed. 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 around 20 to over 40, with most of them closer to 20. Most students in the course were male; only five females were enrolled in each of the courses.

John Lane, 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 fellow graduate student 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.

## Study I Animation Assignments

Winter quarter marked the first time that Professor Lane had ever included animation assignments in his CIS 315 course; it was truly a pilot venture. Not quite knowing what to expect, he opted to structure the animation assignments according to John Stasko's (1997) recommendations. Through correspondence with Stasko, I secured copies of the assignments he had used in his previous algorithms courses at Georgia Tech. Professor Lane and I then worked together to design three animation assignments that resembled those used by Stasko. Appendix A includes copies of the assignment statements we posted; I briefly summarize them below:

1. *Animation Assignment #1: Flying Logos* (worth 1% of total grade). In order to familiarize themselves with Samba, students were asked to design a Samba animation in which a graphic of their name moved around the screen.

2. *Animation Assignment #2: Divide-and-Conquer or Dynamic Programming Animation* (worth 5% of grade). Students were invited to pick a favorite divide-and-conquer or dynamic programming algorithm. At the time the assignment was issued, both of these kinds of algorithms had already been covered, or were being covered, in class. They were then asked to create an animation for the algorithm they had chosen, keeping in mind that (a) the animation should work for general input, and (b) they would present the animation to their classmates at a later date. They were allowed to work individually, or in groups of two.

3. *Final Animation Assignment* (worth 15% of grade). The final animation project was the same as assignment #2 with three exceptions. First, they were encouraged to work in groups, and no limit on group size was imposed. Second, the assignment was more open-ended than assignment #2; they were invited to pick "one or more non-sorting algorithms that employ one of the general problem-solving strategies covered in the course (dynamic programming, divide-and-conquer, greedy, etc.). We gave them three suggestions: (a) illustrate a proof; (b) illustrate a side-by-side algorithm race; and (c) animate a real-world application. Third, the animations they designed were supposed to go above and beyond those programmed for assignment #2, since the final project was worth three times as much. Other than that, the assignment requirements were the same as those for assignment #2; they were asked to present the animations to the class, and they were asked to design animations that worked for general input.

In the following three sections, I present the descriptive part of the study. In particular, I describe how students went about the animation assignments, what happened at the presentation sessions, and how the animation assignments impacted the instructor. Based on the descriptive account, I then take up the two research questions posed by Study I.

## How Students Went about the Animation Assignments

> **Scribbly sketches with colored pencils; basic number crunching to get proportions, units scale; reading documentation to find out how to do what I want; creating hacks to make Samba able to do basic things I want (i.e., rotate, color); roughing out the big parts; details, details, . . .and this goes on for a long time. [I spent] enough [time on this assignment] that I'm too embarrassed to say [how much time I actually spent].**
>
> **(Tim's account of how he did his animation assignment)**

For the final two animation assignments,[63] students animated a total of 17 different algorithms or algorithm themes. For assignment #2, 29 one- and two-person groups completed projects animated 11 different algorithms. On the other hand, 26 student groups, each containing one to four students, animated nine different algorithm themes for assignment #3.[64] The algorithms that students animated, along with the number of groups that animated each algorithm, are presented in Figure 59 and Figure 60. As the figures illustrate, the QuickSelect algorithm was particularly popular for assignment 2, whereas Dijkstra's algorithm, Kruskal and Prim's minimum spanning tree algorithms, and graph searching algorithms (breadth-first and depth-first search) were popular themes for assignment #3.

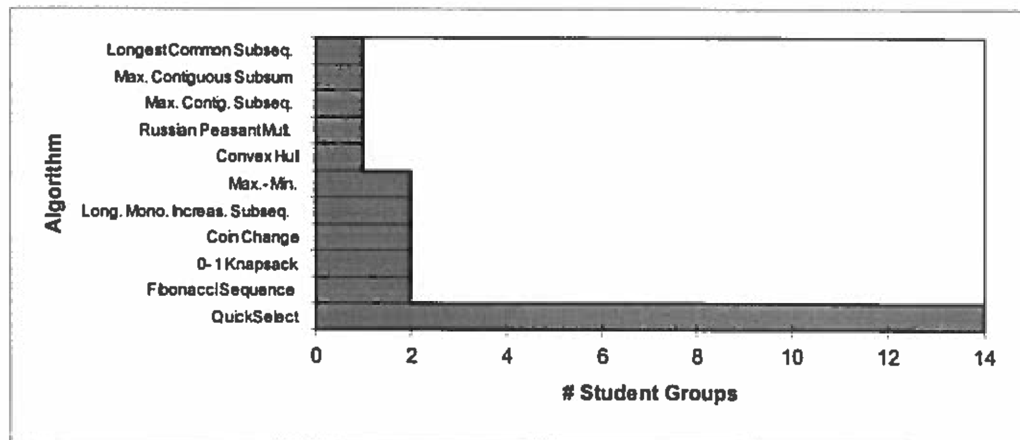



Figure 59. The Assignment 2 Animation Projects

---

[63]I elected not to pursue fieldwork on the Flying Logos assignment.

[64]I say "algorithm themes" because one option students had for assignment 3 was to compare a group of algorithms that solved the same problem.

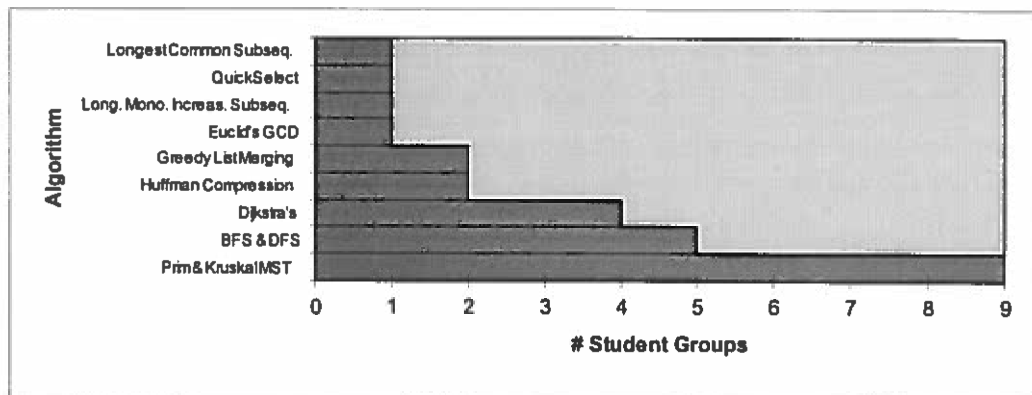



Figure 60. The Assignment 3 Animation Projects

Student surveys, interviews, and participant observation sessions enabled me to gain greater insight into students' animation-building efforts. In the following two subsections, I provide estimates of how much time students spent on the assignments, and I describe the particular animation-construction activities in which they engaged.

## Time Spent

Students dedicated significant amounts of time to the final two animation assignments. According to the survey I administered after the second assignment, students spent an average of 15 hours on that assignment. The results of the survey should, however, be placed in proper perspective. First, the number of students who responded to that survey was low (n=9). Second, variability in the responses was high; one student spent 40 hours, whereas another spent 5. Third, two of the nine respondents spent so much time on the assignment that they elected not to report exact numbers; if their numbers had been included in the data, the average would have certainly increased—perhaps substantially.

With respect to the final animation assignment, my interviews and observations[65], along with students' comments during the final animation sessions, indicated that students generally spent even more time on that assignment than they did on assignment #2. That should not be surprising, given that it was worth three times as much as assignment #2.

## Animation-Building Activities

According to their responses to the survey I administered after assignment #2, students spent

- an average of 20 minutes selecting an algorithm to animate;
- an average of 9.5 hours implementing the algorithm; and
- an average of 5 hours animating the algorithm.

---

[65]Because of the relatively low response rate I received for my first survey, I elected not to solicit time estimates in my survey on the final animation assignment.

Participant observation sessions and interviews enabled me to gain even further insight into what each of these activities entailed; the taxonomy of **Error! Reference source not found.** provides a more detailed decomposition. In the following three subsections, I elaborate further on activities in this taxonomy.

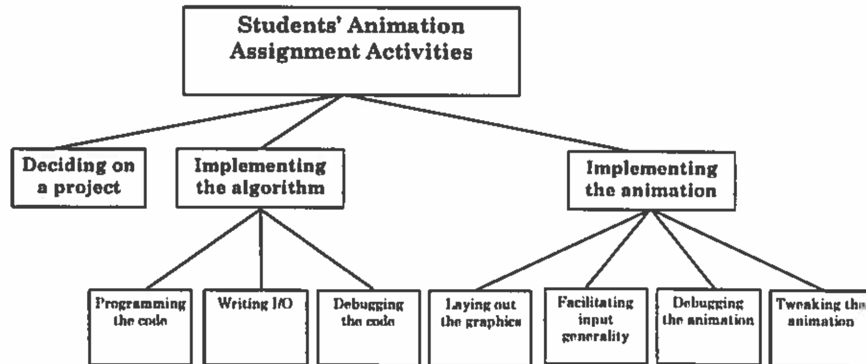



Figure 61. A Taxonomy of How Students Spent Their Time on Animation Assignments

Deciding on a Project

The animation assignments gave students a lot of leeway in terms of the projects they could take on. While some students relished the freedom they were granted, others were stymied by it. As the algorithm animation teaching assistant, I interacted with a handful of students who "had no idea what to do," as one student (Mark) put it.

Further, since Professor Lane did not know what to expect when he assigned the animations, the list of requirements included in the assignment statements was rather vague (see Appendix C). As Mary stated in one of her survey responses, "Just deciding on which were the important parts to get done was a problem. Lots of people didn't quite know what he wanted to be turned in, and it wasn't until afterwards that he figured out what he would have rather seen."

Implementing the Algorithm

Professor Lane did not supply students with 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. In the survey cited above, in fact, algorithm programming consumed, on average, the largest portion of the overall time spent on assignment #2.

In order to facilitate input generality, students had to write routines that read their algorithms' input data from a file. Because some algorithms accepted complex input (e.g., graph algorithms required a specification of a graph), students sometimes spent several hours implementing sophisticated file parsing routines.

Finally, students had to debug their algorithms. As it turned out, students' efforts to debug their *algorithms* were intricately entwined with their efforts to debug their *animations*; they often deferred the task of debugging their algorithms until they had implemented their

animations. I shall discuss students' integrated debugging process in greater detail in the following subsection.

Programming the Animation

Animation programming entailed 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; the lower left-hand corner of the screen is 0,0, and the upper right-hand corner of the screen is 1,1. 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. When this trial-and-error was for the purpose of putting the finishing touches on an animation, it was called *tweaking*, as I shall discuss below.

Second, 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—a process that, as mentioned above, was intimately entangled with debugging their algorithms. Indeed, in the case of the Samba animation assignments, debugging took on a new twist, since observed problems in students' animations could have had one of two causes: (1) a bug in the *underlying algorithm*, or (2) 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 I collected on Mary and Mark indicate that students actually used their animations as a resource for debugging their algorithms. Consider, for example, Mary's description 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.

Similarly, Mark noted that his Samba animation revealed a bug in his graph algorithm:

> I did find one thing, though—that Samba actually helped me debug my program, because I thought it was right, and I found at least two or three more [bugs]. I was getting the right answer, but I was checking the wrong vertices, and I thought that the Samba animation was wrong, and it was actually the algorithm. . . .It let me know that *something* was up, so that was kinda nice.

Nonetheless, as Mark suggested in the above quote, students always had to be wary of what their animations showed them; any apparent error in the animation could be caused either by an error in the animation, or by an error in the underlying algorithm. For some, at least, the process of tracking down the source of an error ultimately became routine; as Mary put it, "you get a real feel for the pattern of it."

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 enjoyed the process. Mary, for example, noted in our interview that, "from an artist's point of view, I really like to be able to tweak the little things, . . .to get it to work out just right, to look neat." In one of her survey responses, she wrote:

> 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.

Further, the lengthy compile-run-run cycle (the first run generates the Samba trace; the second run allows one to view the animation in Samba) required to test a modification to a Samba animation made for potentially long tweaking sessions, as Mary pointed out:

> 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 informant, Tim, had a similar experience. He confessed 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.

## The Animation Presentation Sessions

During five animation presentation sessions held near the end of the term, students were required to present their final animations to Professor Lane, me, and interested students. These sessions lasted roughly one to two hours each; the individual presentations within each session lasted between five and twenty minutes. Despite the wide variance in presentation length, presentations tended to contain the same key features, which I discuss below by way of an example. A few of the presentation sessions turned into open discussions in which students, Professor Lane, and I considered ideas for improving future animation assignments. I conclude this section by discussing the major themes of such discussions.

### A Typical Presentation Session

Table 16 presents excerpts from a content log[66] of a typical presentation session, in which two students (Joe and Jeff) presented their animation of Prim's minimum spanning tree (MST) algorithm. The excerpted content log highlights six events that were common throughout the presentation sessions.

---

[66]Following Jordan and Henderson (1995), I use the term *content log* to denote a timestamped list of noteworthy events. As part of my post-hoc videotape analysis, I created a content log of each presentation session. Note that the timestamps in column 1 indicate the actual elapsed time from the beginning of the videotape.

First, Joe and Jeff started by providing some background on what they did. At counter 14:40, Joe explained that they had devised an illustrative scenario[67] in which a construction company has to wire all houses in a neighborhood using the least possible amount of cable. Treating the neighborhood as a graph in which vertices are houses, and in which edges are roads, the company proceeds to use Prim's algorithm to lay the cable. Students' presentations typically started with this kind of introduction, which explained what algorithm was being animated, and the particular approach that was used to illustrate it.

Second, it was not until over two minutes after Jeff first started the animation that we finally saw the initial graph screen. Indeed, it often took students a while to load their animations (counter 15:10), to arrange the windows on the screen (counter 15:20), and to step through the animation until it reached its logical starting point (counter 16:40). Needing two minutes for this process was not unusual.

Third, the process of stepping through an animation entailed two complementary events. At various points in the animation, they provided play-by-play narration (see counter 16:40, 21:30). At other points, they opted to speed the animation up, since they thought that what was happening was not of interest (see counter 20:15; 21:45). The practice of providing step-by-step narration of interesting events (e.g., when an edge is not selected, because it would form a cycle), and the practice of fast-forwarding through uninteresting events (e.g., the routine selection of an edge at the head of the priority queue) turned out to be typical throughout the presentation sessions.

---

[67]Scenarios were a notable feature of several student animations in Study I and Study II. See Appendix B for an in-depth look at students' use of scenarios.

Table 16. Excerpts from the Content Log of a Typical Presentation

| Counter | Description |
|---|---|
| 14:12 | Presentation begins. They change to directory in which their animation resides. |
| 14:40 | Joe introduces the animation: "We implemented Prim's algorithm to. . .[lay] cable between houses, to see what the shortest amount of cable you would need to do that." |
| 14:55 | Joe discusses a major challenge: "The real trick in this was getting the graph all set up. . . .In the book, it said, 'give me the edges, and give me the vertices. Well that came out to 600 lines of code." |
| 15:10 | Jeff starts animation. |
| 15:20 | Jeff starts arranging the windows of the animation |
| 15:50 | Jeff confides that they wrote a total of 3,000 lines of code to implement the graph graphics. They implemented a vertex class, an edge class, an adjacency list class, etc. |
| 16:40 | Jeff begins stepping through the animation. An explanation window on the right explains what's happening in the main window. A priority queue window is also on the right. |
| 18:25 | First house turns red, to indicate that it has been wired. Tank moves down to the house. First house's adjacent edges thicken, to indicate that they are under consideration. Jeff and Joe continue with the narration. |
| 19:15 | Jeff comments that they had a function that flashed edges on and off, but that they couldn't use it because it caused a segmentation fault. Joe comments that when they tried to alter text, it would core dump. Professor Lane asks me to write these comments down for the benefit of Stasko. Joe requests that I also write down that he doesn't want the control panel to show up behind other windows by default |
| 20:15 | I ask whether there's color-coding being done in the queue window. They say that the queue view does not have the same color coding that's on the graph. |
| 21:02 | Jeff fast-forwards the animation to the next time when "it makes a circuit." The animation advances rapidly, and then Jeff slows it down, and points out that "we took the example right out of the book," but that "we did it a little differently from the book." |
| 21:30 | Jeff continues narration: "Now it's getting the shortest path, which turned out to be six, back to the main house (the first house added), checks to see if it's already wired, then it deletes the edge [because the house is already wired], and tries the next one." |
| 21:45 | Jeff comments that "from here on out, I can just speed this thing up." He proceeds to fast-forward the animation to the end. |
| 21:55 | Animation finishes. It reports that they only used 37 miles of cable. I comment that "you saved a lot of money," and Joe adds, "minus the expense of the tank." We laugh. |
| 22:20 | John asks how to prepare an input data file. Joe and Jeff proceed to explain how to do that. You have to include coordinates of the vertices in the graph. |
| 25:15 | Their explanation of the file format ends. |
| 26:05 | John asks whether they had any other ideas about the priority queue. Jeff responds that they "should have drawn it as the tree that it is." |
| 26:15 | I suggest that they could have coordinated the edges in the queue with the edges in the graph. |
| 26:25 | Joe confessed that they added the priority queue illustration in only about an hour last night at the end. I pointed out that it ended up being just a "text dump," and they commented that they had planned to do something nicer, but that they "sort of ran out of time." |
| 26:45 | Presentation ends. |

Fourth, at various points in the animation, Joe and Jeff took opportunities to describe their implementation difficulties and accomplishments. Joe explained that they needed 600 lines of code to implement a simple one-line instruction in the book's pseudocode (counter 14:55). Later on, Jeff described a 3,000-line graphics library consisting of general-purpose vertex, edge, and adjacency list classes (counter 15:30). Finally, Joe and Jeff reported a couple of Samba bugs they encountered, and provided a suggestion for improving Samba (counter 19:15). Given that students spent substantial amounts of time on the implementation of their animations, it is not surprising that they wanted to share their accomplishments and frustrations. Like Joe and Jeff, many students took the opportunity in their presentation sessions to do so.

Fifth, I broke in at one point (counter 20:15) in order to ask a question about one of their animation windows: whether or not the priority queue window uses the same color-coding as the graph window. John and I frequently broke in with such questions, which served to clarify the meaning and significance of students' animations.

Finally, at the end of the presentation, Professor Lane and I provided suggestions on how they might improve their animation. By asking a question, Professor Lane implicitly suggested that they might consider an alternative illustration of the priority queue (counter 26:05); Jeff's response indicated that he was well aware of the animation's weakness. I followed up John's question by commenting that they might have better coordinated their priority queue illustration with their graph illustration (counter 26:15). In response, they confessed that they ran out of time at the end, and had only about an hour to write the priority queue illustration.

Such suggestions for improving an animation typically came at the end of the animation presentations. Some suggestions were one-sided monologues; Professor Lane did the talking, and student presenters did the listening. As was the case in Joe and Jeff's presentation, other suggestions often stimulated collaborative design discussions, in which the students themselves followed up with their own suggestions for improving their animations.

## Assignment Improvement Discussions

As illustrated by Joe and Jeff's presentation, students were often aware of their animations' weaknesses; they had in mind to do something better, but simply ran out of time. In several animation presentations, students openly voiced frustration over the animation assignments, and made suggestions for improving them. In a few cases, students' frustrations and suggestions stimulated lively discussions, in which John, student presenters, and I considered ways of improving various aspects of the assignments. As discussed in the following three subsections, such discussions revolved around three key themes.

### The Value of Programming the Algorithm

In their final presentations, some students questioned the value of having to write their own algorithms from scratch. For example, after he presented his animation, Kyle suggested to Professor Lane that it might be helpful to give students a set of pre-programmed algorithms:

> If somebody had written, let's say four of the algorithms that we covered in class, and you gave those to us to animate. That way, we could just work on the actual animation of the algorithm, have it like already done and all correct.

This comment compelled me (E)[68] to ask Kyle (K) whether he felt he learned anything from implementing the algorithm. The conversation continued, with John (J) breaking in at the end:

E: Well, did you learn anything in implementing the algorithm?

---

[68]I shall use "E" (the Ethnographer) to denote myself, and "J" to denote Professor John Lane, in all vignettes that appear in this and the next chapter.

K: I learned a lot of C++ in implementing the algorithm, and I learned a lot about C++ templates and overloading operators. I'm not sure that had anything to do with the algorithm itself.

J: So, you didn't illustrate the data structures, and you didn't illustrate the C++ in any way.

K: Exactly, I did this entire thing with my data structures book from 313.

J: You know, that's interesting. It could be—I have my doubts about C++ in general—for Samba, in particular, this doesn't seem to be a very good language for basing the Samba programs in.

As this sequence illustrates, Kyle's concern was that C++ programming is a diversion from CIS 315's objectives. Professor Lane's follow-up questioning indicated that he was in agreement: Writing algorithms in C++ may not encourage the kind of thinking about *algorithms* that is the emphasis of CIS 315. This insight led Professor Lane to suggest an entirely different kind of language on which to base Samba programs:

> Take a look at the kinds of things we want to do. I mean, you should have a language which is not very different from the pseudocode that might be in the book for some of these things, right? Something much more Pascal-like, and much simpler. And, you don't need very much.

The Difficulties of Graphics Layout

A few students indicated that laying out the graphics for their animations had bogged them down. Consider, for example, the following conversation, which took place after Kyle (K) finished presenting his animation of Dijkstra's shortest-path algorithm. Professor Lane (J) provided some feedback, and I (E) followed up with a question:

K: So that's my algorithm animation. Did it. . .I mean, I tried to make it highlight what the algorithm was doing in some meaningful way. Do you think that it fulfilled [the requirements of the assignment], or?. . .

J: Well, you could sort of see what it is doing. What I don't think is captured, but I'm not sure many people have captured anything like that, is why the algorithm is working.

K: Yeah, I'm happy that it works. . .I'm not exactly happy with what I had to do in Samba. . .I didn't find it very helpful. I mean, I could have done other things. . .

J: Well, there's an interesting point. Some of the problems you have with Samba and so on seem to be so standard that you wonder whether something can't be built into that. . .

E: (To K) So, you find yourself struggling just to get the thing to go, much less. . .

K: Well, I have the actual text version of it—no problem, everything's right. Then, I spent like the weekend doing like Samba fixes, basically. I mean, up until this weekend I'd been working on it constantly, just to get it to even work. . .

E: So what particular Samba problems did you encounter?

K: Well, for one thing, to place the graph, to place the nodes—that was like consuming a lot of my time. . .I tried to do something fancy and then I just quit. And then, just like labeling things so that they were clear wasn't very easy to do either. I mean, I had a lot of problems with that, so I sat there and like worked at that.

E: So you had to get the labels near the things you were labeling?

K: Yeah. . . .I mean, if there was, like, say, a "label this object" [command],. . .that would have saved me a day.

Kyle's frustrations were quickly echoed by Mark (M), another student who was to present his animation of depth-first search after Kyle finished. The conversation continued:

E: [To Mark] You're nodding your head. You think the same?

M: I think exactly the same. . . .It took me a long time just to label the vertices also. . . .I spent most of my time doing mathematical calculations just to figure out where to place the text, and where to place the vertices. . . .

J: That's interesting. These are useful exercises, but not in this class. . .

In the above exchanges, Kyle and Mark clearly indicated that they had spent significant amounts of time on the actual layout of their animations—that is, placing the graphics on the screen, and appropriately labeling them. However, as Professor Lane pointed out, in the process, they appear to have missed out on thinking about the central themes of CIS 315: algorithm correctness and efficiency.

Furthermore, in both cases, the students had their sights set on more ambitious projects, but got so bogged down in the details of graphical layout that they were unable to finish. As Kyle confessed, "I tried to do something fancy and then I just quit." Similarly, as Mark later confided,

> I found that a lot of my time was spent figuring out where to place things. I was hoping to do a breadth-first search and a depth-first search to compare on either side, but I ended up not having enough time. That was my problem.

<u>Input Generality a Formidable Challenge</u>

In Mark's presentation of his animation of breadth-first search, the difficulty of designing general-purpose animations came to light. To satisfy the input generality requirement, Mark opted to write a graph layout algorithm that positions the vertices of the input graph around a circle. Professor Lane's concern over whether such a circle layout does justice to the algorithm prompted the following conversation, in which the professor arrived at a new perspective on the input generality requirement:

J: For breadth first search, laying it out around a circle would not be very instructive, I think. . .[W]hat you'd want to do is actually set it up with some graphs where you would see it—especially with breadth-first search—you'd see it searching out[wards].

E: A hierarchical structure?

J: Well, I don't know. I mean, you think about what breadth-first search is, and what if you were teaching this and going to a board and showing depth-first search working. How would you lay out a graph to see if it's working?

M: I guess I should have restricted it a little more. I was hoping to get more flexibility, so you could have more nodes, you could just type away, and have more nodes. But I guess I should have restricted that down, and made it more structured, so you could see it better. . .

J: You'd be able to see what's happening in all of that. This graph layout thing is a real problem. And, one of the things you have to keep in mind here[:]. . .Having it be able to handle random input is not necessarily very useful. . . . .You're not writing the program for it; you're trying to explain *why* the program works.

In line with the assignment requirements, Mark's interest was in "flexibility"—something that was supported by his circle layout algorithm. Yet, in his effort to make things work in general, he not only had to dedicate significant amounts of time to graphics programming, but also compromised the quality of the animation—at least from the perspective of Professor Lane. In the end, Mark's experiences were as much a lesson for Mark as they were for Professor Lane. As Professor Lane later noted, "I think that's something that I would

emphasize to students in the future: You think about good graphs, come up with some examples which are going to show what's working."

The Instructional Side of the AV Assignments

Professor Lane and I collaborated extensively to carry out the responsibilities entailed by the animation assignments. In particular, we undertook six different activities:

**Installing Samba.** I transferred the Samba software to the Department's network via ftp, and then installed it locally in a directory accessible to all CIS 315 students. The transfer and installation proceeded relatively smoothly; in all, the process took me perhaps three hours.

**Designing the assignments.** Before the term began, and then intermittently throughout the term, Professor Lane and I discussed how to work the animation assignments into the course, and how to phrase the assignment statements. At first we relied on Stasko's assignments for guidance. Like Stasko, we elected to give three animation assignments, and to make them worth roughly 20% of the overall grade. Unlike Stasko, John wanted to give students a lot of flexibility in terms of the specific algorithms they animated; thus, we ended up putting a couple of hours into figuring out how to make helpful suggestions on what to animate, without actually giving "answers" away.

**Discussions with students.** Students periodically approached John and me regarding the animation assignments. They approached John for advice on what to animate, and for feedback on their animation ideas. They approached me for technical advice on Samba, and for suggestions on how to pull off an idea in Samba. While I cannot say for sure how much time John and I spent with students over the course of the term, I can say that we were not overly burdened with students' questions. To the contrary, John sometimes complained that students were not coming in to see him (regarding their animation assignments) frequently enough, and that the infrequency of their visits worried him.

**Obtaining presentation technology.** In order to facilitate student animation presentations, we needed to find a way of projecting a Sun SparcStation screen onto a larger movie screen. The Department did not own such a projection device, and it took me awhile to track down someone who did—so long, in fact, that we had to cancel the presentation sessions for assignment #2. For the final assignment's presentations, another professor in The Department let us borrow a device that hooks into a SparcStation (among other computers), and rests on an overhead projector. Prior to the presentation sessions I spent about an hour exploring its functionality and limitations. Prior to each presentation session, I needed about fifteen minutes to set it up to work in the Colloquium room. While the quality of the projection was greatly inferior to the quality of the SparcStation screen being projected, the device served its purpose adequately.

**Arranging presentation sessions.** All three presentation sessions, each of which lasted between one and one-and-a-half hours, were scheduled outside of regular lectures. John offered several alternative times to the class, and selected times based on when the most students would be able to attend. Those few students who could not attend a presentation session gave John a private showing at a mutually agreed-upon time.

**Grading.** Although he provided me with no specific time estimate, I believe it is fair to say that John spent a considerable amount of time grading the animation assignments.[69] Once he figured out how to grade the assignments (an endeavor that was not without its problems, as I shall discuss shortly), he ran and evaluated, one at a time, the actual Samba animations that students had handed in. Instead of constructing his own input data for the animations, he opted to rely on the students' canned examples (which they were required to hand in along with the general-purpose programs that generated Samba animation files). In grading the final animation projects, he used my videotapes of the animation presentations to guide his grading; as he viewed each presentation on videotape, he brought up the corresponding animation on his Sun SparcStation and tried it out himself.

## Assessing the Costs and Benefits of AV Assignments

In this section and the next, I revisit the research questions posed by this Study In light of the observations reported in the previous sections. Recall that the first research question was, What are the costs and benefits of AV assignments? To address this question, I consider both the students' and the professor's perspectives.

### Students' Perspective

The descriptive account of students' presentation sessions offered in the previous section foreshadowed students' general impressions of the costs and benefits of the assignments. On the one hand, my fieldwork suggests that students generally enjoyed the assignments. In students' survey responses, the most frequently cited benefit of the animation assignments was their hands-on nature; they were valued as a concrete task in a class that was otherwise theoretical and abstract.[70] As Mary wrote,

> This assignment was long and involved, but rewarding in its own way. I liked being able to make classes that handled Samba code, and then fitting them together like clockwork. I liked using principles from a graphics class I took, and using math to work out a smooth, pleasing way to represent a variety of data input and output sizes.

Similarly, Tim commented on the value he derived from using his own concrete animations to learn about abstract algorithms:

> I like that building an animation is a concrete task, is achievable, when you are done you have something. . .In this course much of what is being taught can be difficult to assimilate. . . .When I build an animation I need to set up landmarks which, because they are objects of my own design, make sense to me so I do not get as lost. Because I am building something I do not get bored; because I can play it back, I can get what was missed during construction.

A second important benefit of the assignments was that they gave students the freedom to be creative and innovative; indeed, many students relished the opportunity to come up with ways of illustrating algorithms. For example, during an early animation help session that I

---

[69]To gain further insight into the grading process, I had planned to observe John as he graded the assignments. As it turned out, his grading style did not allow for that. He tended to use free scraps of time to grade assignments; he was reluctant to commit to a definite grading schedule, which might have allowed me to come in and observe him.

[70]Note that this finding echoes that of Stasko's (1997) experience report.

led for Professor Lane, I observed a group of three students engaging in an enthusiastic brainstorming session. Intent on coming up with clever ways to depict the algorithms they were to animate, they were visibly excited as they devised and discussed stories of grocery shopping and giving change at a supermarket.

On the other hand, the animation assignments left many students frustrated. Students' frustration had two causes. First, students found themselves unable to implement, in their entirety, the original animations they had in mind; there was simply too much to do in too little time. In our interview, Mary succinctly summarized the problem thus:

> I guess the problem was that there's just too much to do. To get everything where you want to be—I mean, everybody in that class had some idea of what they really wanted to do, and everybody felt like they couldn't get it all done.

Second, students often felt that the final animations they handed in belied the effort and thought they had put into them. For example, in her response to my first e-mail survey, Carol vented her frustration over not being able to tell the story behind her animation:

> Believe it or not, I spent a lot of time [on this assignment]. However, seven hours before it was due, I realized I would never make things work, and I changed the whole thing to just make one that does something with numbers. I totally ran out of time, and my final animation was so sloppy. It disappointed me a whole lot. I spent a lot of time, but my outcome was really sloppy, and you and other graders will have no idea of how much I struggled and how much time I spent on it by just looking at the animation I turned in.

Similarly, Mary wished that she had had the opportunity to hand in a document describing the reasoning behind the design decisions her group had to make:

> It would probably be helpful to say what we thought was important about [our animations]. . .I [would have wanted to] mention things about our *design* considerations, why we chose *not* to represent certain things graphically, why we did. And, it would have nice to turn in something like that, so the professor could have seen how hard it was, what we went through.

According to students, then, their finished animations inadequately reflected what they had actually done; they wished that Professor Lane could have somehow taken into consideration all of their time and effort.

### The Professor's Perspective

Professor Lane's emerging perspective on the costs and benefits of the animation assignments was not only foreshadowed in, but shaped by the "assignment improvement" dialogs presented above. By the end of the term, it was clear to me that John was generally sold on the idea of algorithm animation assignments. He liked the new twist they added to CIS 315, which he had taught without such assignments for 12 years. Moreover, I observed several instances in which students approached him to discuss algorithm animation ideas, and John got visibly excited. He obviously enjoyed thinking about and discussing ways of graphically illustrating the algorithms covered in the course.

John made it clear that there was much room for improvement. In my interviews with John, we discussed the problems with the assignment at length. In the following two subsections, I consider John's concerns over what he did, and what his students did.

Grading is Difficult and Takes Too Long

John had trouble figuring out *how* to grade the animation assignments. He had in mind that students would establish a "contract of expectations" with him prior to handing in the assignment; however, few of them did. As he noted in our end-of-the-term interview, the lack of such a contract made things difficult:

> If you look at the differences in the projects, it was night and day in some instances. But you take something that was very poorly implemented, but students put in some effort, they illustrated an algorithm, I don't think there was anything creative behind it, but how many points can I really deduct for all of this?. . . Given that we [did not have] some sort of agreement—a contract about the expectations—. . .I think the grading is very difficult.

Once he had made decisions regarding his criteria, it did not take much time to go through and do the grading. As he put it, "It's a lot of time to make the decisions. It's not really a lot of time to go through the animations when you sit there."

He did, however, point out that he experienced difficulties in figuring out how to run students' animations as they were meant to be run—that is, with the windows configured properly, and at an appropriate speed:

> Things weren't working the ways students said they would work; when you have a lot of windows appearing all over the screen, you don't know what to move where, so that you to run the thing twice. You don't really know what speed to run it at. Sometimes you run these things at slow speed, and you wait a half an hour, nothing's happening. So, you have to run it fast. But, if you run it fast, sometimes it runs through too quickly. So, there's no real good warning about that.

For both assignments, students were supposed to submit a README file documenting how to prepare an input data file. While some students prepared helpful README files, many did not. As John noted, "Sometimes it was impossible to understand what they meant, how to prepare a data file. . .I tried several of these things; they didn't work as students specified."

Students' Time Not Well-Spent on the Animation Assignments

With respect to the ways in which he saw students spending their time on the animation assignments, John repeatedly voiced three distinct but closely-related concerns in our discussions and interviews. The first stemmed from his observation that the amount of time students were spending was incommensurate with the overall point value of the assignments. After students had handed in assignment #2, for example, John and I talked about this:

> J: They're very wrapped up in the programming—even more so than in typical programming assignments, because, understandably, they're very excited about what they're doing. And the amount of time they put into this for what is five percent of their grade is very much out of proportion to the amount of time they, for example, put into homework, or even studying for the exam. . .
>
> E: So, where's the time going?
>
> J: Well, I'm not sure. Some students claimed as much as 15 hours for these five points of the grade. And fifteen was when I stopped asking. I mean, some of them have even gone beyond all of this. Now, an exam is worth 20 points. I don't think anyone would claim they'd studied 60 hours for an exam.

Likewise, when I interviewed John after students had handed in their final animation projects, John remarked that "students spent far too much time on the project overall."

A second concern of John's was that the time students were spending on the animation assignments detracted them from other important course activities, including doing homework problems, attending lectures, and studying for the final exams. As he saw it, the time they spent on their animations "had to come out of something." In the following interview passage, he cites evidence that the time was coming out of these other activities:

> I know from just finishing grading the final exam that it's quite obvious that a lot of them did essentially no studying for the final. . . .At the time of the final, . . .they were spending those last couple weeks getting their animations to work. . . .They weren't doing their homework. . . Attendance was down those two weeks. . . .[T]hey all said, at the time of the final, that they were spending those last couple weeks getting their animations to work.

However, while he did not necessarily like it that students were spending all of their time on the animation assignments, he sympathized with their situation:

> And [an animation] was something you just can't [abandon]. . . .You've already put a lot of effort into the animation; it's not as if you can say, "Well, the hell with it, the final's worth more points." So, at this point, you're gonna say, "Well, the final, I don't know what's going to be on there; maybe I have to study, maybe not; maybe I study, and it doesn't do any good. But I *have* to get this animation in, because I know I'll get the points for that."

John's third concern, closely related to his second, was that students were not spending their time on the "right" things when they worked on the animation assignments. Recall that the main objective of CIS 315 is to argue formally about why algorithms work, and to talk precisely about how efficiently they work. John's impression was that the students did not grapple with those issues enough in the course of constructing their animations. Instead, as discussed above, students spent most of their time steeped in the minutiae of programming a general-purpose Samba animation: laying out the graphics, building animation classes, getting the animation to work, tweaking the animation's appearance, etc.

While admittedly an extreme case, one student's experiences with assignment #2 well illustrates the nature of John's concern that students were not spending their time on the right things. For assignment #2, Fred chose to use the metaphor of a forest ranger in a forest to illustrate a divide-and-conquer solution to the max-min problem.[71] In Fred's animation, the forest ranger's objective was to measure, as efficiently as possible, all of the trees in the forest, and then to report the height of shortest and tallest trees. Fred proceeded to develop an attractive animation in which the forest ranger employed what Fred thought to be the divide-and-conquer method discussed in class. In particular, the forest ranger divided the forest in half; found the tallest and shortest trees in each half (by walking through the forest with a measuring stick); and finally merged the solutions.

What Fred failed to realize was that his animation did not take the divide-and-conquer strategy far enough. Instead of recursively dividing up sub-forests until sub-forests of size two were reached, Fred's animation prematurely cut off the recursion after the forest had been divided just one time. As John noted in an interview shortly after Fred had come in to talk to him,

> [Fred] got very wrapped up in [programming the animation] very quickly, and he did not understand what algorithm he was programming, and wound up not programming the algorithm

---

[71]For a description of the problem and the divide-and-conquer solution, see (Cormen, Reiserson, & Rivest, 1990, chapter 10).

> at all. All he did was think about dividing forests in half, and so on. He missed the whole point, so this was a very interesting animation, and he thought a lot about that, but he didn't really think about the algorithm.

To John, Fred's case warned of the potential dangers of assignments that require students to engage in activities far removed from the focus of the course. As he put it, such assignments open up the possibility for students to animate "superficial details," without "really showing what the algorithm is all about."

## Why are AV Assignments Effective?

The second research question posed by this study aimed to address the need for an alternative theory of effectiveness for AV artifacts: Why are AV assignments effective learning exercises? Plainly, this question depends intimately on another question: What does it mean to be an *effective* learning exercise? These two questions are central to this research, and are taken up in earnest in Chapter V. Below, I sketch out preliminary answers—ones that are suggested by the fieldwork I undertook within the scope of Study I, and that guided my subsequent fieldwork in Study II.

To a first approximation, an *effective* learning exercise in CIS 315 should help students to develop one or more of the skills that the three course objectives define:

1. to teach students efficient algorithmic techniques for solving various classes of interesting problems;

2. to teach students how to analyze an algorithm's efficiency; and

3. to teach students how to construct convincing proofs of an algorithm's correctness.

While they likely imply much more, objective (1) minimally implies an ability to explain the procedural behavior of the various algorithms of interest in the course; objective (2) minimally implies the ability to take a specification of an algorithm and provide the number of steps it requires to execute as a function of its input size—a quantity that is expressed using Big-O notation; and objective (3) minimally implies the ability to communicate an algorithm's correctness with the help of various established proof techniques covered in the course, including loop invariants and induction.

Notice that all of these are proficiencies that one would expect an *algorithms teacher* like Professor Lane to possess. Indeed, an algorithms teacher would not be an algorithms teacher without these skills; they both define, and are *prerequisites* for, someone in that position.

It follows, then, that learning activities that provide students with opportunities to do the kinds of things that an algorithms teacher does—for example, talking about interesting algorithmic techniques, performing an efficiency analysis, or talking through a proof of correctness—would be potentially effective. Moreover, such learning activities would do well to provide opportunities for algorithms teachers not only to access students' emerging competence, but also to interact and give feedback to them regarding that competence. Indeed, how could an algorithms student possibly know that her or his performance is on track, how could an algorithms student possibly improve her or his performance, without such feedback and interaction?

It is in these two senses that the AV assignments given during Study I appeared to be effective. On the one hand, the assignments offered students crucial opportunities to "do as a teacher does." Professor Lane came to realize this key benefit as the term progressed:

> 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 [constructing animations].

From John's standpoint as the course instructor, having students teach was a crucial evaluative tool, 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 teach meant having to take seriously what was important about the algorithms they were teaching—an endeavor for which animation construction appears particularly well-suited. Consider, for example, the following interview sequence, in which Mary cites 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.

Furthermore, in a quote already presented, Tom cited two additional benefits of animation construction as a means of coming to grips with what one has to teach: (1) self-constructed AVs are "objects of [one's] own design," so they make sense to the person doing the construction; and (2) self-constructed AVs require one to engage in a creative process, so that one does not get bored.

On the other hand, the final animation presentations provided students and Professor Lane with crucial opportunities to engage in meaningful discussions regarding students' emerging competence as teachers. 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 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 performed 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. The creative thought that they put into their animation designs, and the hard work that they put into their animation implementations—both of these seemed to vest students in the teaching activities that they were undertaking. In turn, they seemed particularly open and responsive to the feedback they received during their presentation sessions. A testament to

that openness and responsiveness can be seen in the extent to which they participated with Professor Lane and me in discussions of ways in which they might improve their animations. As the example of Joe and Jeff's presentation illustrates, students seemed particularly interested in engaging in such discussions—in listening to, and offering their own reflections on, how they might "do better" as teachers.

## Discussion

The student-constructed AV assignments proposed by a few computer science educators violate the flow of knowledge from expert to learner presupposed by EF Theory; hence, EF theory cannot account for any pedagogical benefits they might have. Past experience reports suggest that they are, indeed, valuable pedagogical exercises; however, those reports have been stymied by the position of advocacy taken by their authors, who have invariably been the very same people who created the AV tools used by the students in their reports. Moreover, the limited range of data on which those reports have been based (observations of course instructors, and occasionally student surveys) has prevented them from furnishing a satisfying account of AV assignments' costs and benefits.

By using a wide range of ethnographic field techniques, and by assuming the dual-perspective of both students and the instructor, the study reported in this chapter was able to furnish a richer descriptive account of student-constructed AV assignments than those offered in previous experience reports. As we have seen, students spent substantial amounts of time on the final two animation assignments. Their time was divided among a several activities, including programming and debugging the algorithm to be animated, laying out their animations, writing general-purpose graphics routines, and debugging and tweaking their animations (see Figure 61). A series of animation presentation sessions at the end of the term afforded students the opportunity to present their final animations to their instructor and peers. Aside from sharing their animations and discussing how they implemented them, students raised several concerns regarding the animation assignments during those sessions. Their concerns sometimes sparked lively discussions, in which student presenters, Professor Lane, and I considered ways of improving the assignments. On the instructional end of the animation assignments, Professor Lane and I spent time designing the assignments, arranging students' animation presentations, obtaining the technology necessary for those presentations, talking with students about their assignments, and grading the assignments.

Second, my fieldwork enabled me to gain insight into the benefits and costs of the AV assignments from the perspective of both students and the instructor. In a class that emphasizes the abstract, students liked the fact that the animation assignments were concrete entities; as Tim put it, "when you are done, you have something." In addition, students seemed to take great pleasure in coming up with ideas for their animations; however, they often became frustrated when they found that they did not have enough time to turn those ideas into working animations. If their final animations did not meet their expectations, students worried that they would not receive due credit for the large amounts of time and effort they had to put in to the assignments.

Although Professor Lane was generally enthusiastic about the animation assignments, he was not altogether happy about the amount of time students spent on the assignments, and the ways in which students their time. Laying out the graphics for an animation, making the animation work for general input, debugging the animation, and tweaking its final appearance—these activities, according to Professor Lane, not only consumed too much of

students' overall time on the course, but were also peripheral to the focus of the course, ultimately *distracting* students from that focus. Furthermore, ambivalence over how to grade the assignments set back Professor Lane's grading efforts; he ended up spending more time on grading than he would have liked.

Third, Study I fieldwork suggested a provisional explanation of why student-constructed AV assignments might be effective. They provide opportunities for students to participate as instructors in the course. By constructing animations of how, why, and how efficiently an algorithm works, and by presenting such animations to the class, students engage in many of the same activities that instructors typically undertake. In addition, student-constructed AV assignments that include presentation sessions (which were not included in the assignments described in previous experience reports) provide students and instructors with opportunities to engage in mutually-meaningful discussions regarding algorithms. During the course of such discussions, instructors are able to give students crucial feedback on their performance as teachers.

Finally, Study I pointed out at least three serious problems with the animation assignments in their current form: (1) students spent too much time on the assignments; (2) the time they spent was not spent on the "right" things; and (3) they did not interact with the instructor enough and early enough regarding their animations.

# APPENDIX B

## ETHNOGRAPHIC STUDY II

> Based on the storyboarding technique introduced by the animation industry in the 1930s, *visualization storyboarding* takes advantage of the familiarity and dynamism of art supplies and human conversation to create a situation in which humans can describe computer programs naturally.
> (Douglas, Hundhausen, & McKeown, 1995)

For the next term of CIS 315 (spring term), Professor Lane maintained an interest in the algorithm animation assignments, but sought to overcome the problems encountered in Study I. In this appendix, I report on Study II, which considered Professor Lane's and his students' experiences with a substantially revised animation assignment during the subsequent offering of CIS 315.

I begin by discussing the revised animation assignment. Next, I introduce a revised set of research questions that drove my Study II investigation. My field techniques for addressing those questions, which differed only slightly from those of Study I, are then covered. I then turn to discussions of my key findings with respect to the revised set of research questions. Finally I considers the implications of both Study I and II for an updated version of sociocultural constructivist theory.

## Study II Animation Assignments

To overcome the problems encountered in Study I, Professor Lane and I opted to revise the animation assignments for Study II in three significant ways. First, to reduce the amount of time students spend on the assignments, we decided to drop two of the three animation assignments for the subsequent term; only a final animation assignment, worth 15% of the overall grade, was retained. By giving only one animation assignment, we hoped that students would not feel as overwhelmed as they did during Study I, and that they would be able to focus more intently on a single animation project.

Second, to encourage students to spend time on the "right" things, 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. While they were free to generate their animations from implemented algorithms, they were not required to do so; writing Samba trace files by hand was equally acceptable.

Third, we hoped to foster increased student-professor interaction by turning 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 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. At those presentations, students would have the opportunity to ask questions, and to receive valuable feedback and suggestions from their professor and peers. 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. Aside from the fact that they were not required to implement general-purpose animations, the "Samba animation" phase of the assignment was essentially the same as each of the final two animation assignments of the previous term.

## Research questions

The main research question posed by Study II followed directly from Professor Lane's interest in overcoming the problems encountered in Study I:

> Is the revised AV assignment format an improvement over the Study I format?

Based on the three changes made to the assignment, this general question could be reduced to three specific research questions:

3. Is a single animation assignment an improvement over three animation assignments?
4. Is the absence of an input generality requirement an improvement?
5. Is the addition of a storyboard phase an improvement?

Given the specific problems that each of the three assignment modifications purported to overcome, the definition of "improvement" could, for each of these questions, be framed in terms of more specific questions:

> *Question 1.1:* Are students dedicating less overall time to the assignment? Are they better focused on their animations?
>
> *Questions 1.2 and 1.3:* Are students spending less time on implementation details (e.g., graphics layout and labeling, graphics library creation, debugging), and more time thinking about algorithms? Are students better able to implement what they have in mind for the assignment?
>
> *Question 1.3:* Are students interacting more with the professor regarding their animations? Are students' animations "better" from the professor's perspective—that is, do they better address the issues important to the course? Are the assignments easier to grade?

Besides Questions 1.1, 1.2, and 1.3, the addition of a second (storyboard) phase to the assignment raised a second set of questions regarding the merits of each individual phase of the assignment. Specifically, I was interested in comparing the storyboard component of the new assignment with the Samba animation component of the assignment along several dimensions:

2.1. How much time do students put into storyboards, as compared to Samba animations?

2.2. What do students learn from constructing storyboards, as compared to constructing Samba animations?

2.3. To what extent does what students learn from each of the two phases of the assignment resonate with the objectives of the CIS 315 course?

2.4. Are both phases of the assignment necessary? Might one or the other be eliminated?

This second set of questions was more exploratory, and aimed at further refining my evolving theory of AV effectiveness.

## Field techniques

For Study II, I maintained my dual role as student observer and volunteer teaching assistant for algorithm animation. Likewise, my fieldwork techniques remained largely the same. One notable exception was that I elected to drop e-mail surveys; I did so because of the relatively low response rate they had garnered during Study I. In their place, I sought a technique that would allow me to access a broader sample of students. Ultimately, I convinced Professor Lane to make *student diaries* part of both phases of the animation assignment. The only catch was that, while a thoughtful, well-kept diary could help a student's grade, a poorly-kept diary could not hurt a student's grade. Because the thrust of the assignment was algorithm animation, we did not want to concede any more of the overall grade to a diary.

To offer students guidance on what to keep track of, and to increase the chances that they would actually keep track of what they did, I supplied them with a standard diary form (see Appendix C). For each work session, the diary form provided space for them to record how many hours they spent, what activities they undertook, what problems they encountered, and any additional comments they had.

Table 17 summarizes the particular fieldwork techniques I employed to explore the research questions posed by Study II. The following two sections describe students' animation assignment activities, as well as the storyboard and final animation presentation sessions. In light of those observations, I then consider the two groups of research questions.

Table 17. Field Techniques Used to Address Study II's Research Questions

| Research Question | Field Technique(s) used to Address Question |
|---|---|
| 1.1. Is having a single animation assignment, as opposed to three, an improvement? | Participant observation, diaries, interviews |
| 1.2. Is the absence of an input generality requirement an improvement? | Participant observation, interviews |
| 1.3. Is the addition of a storyboard phase an improvement? | Participant observation, interviews |
| 2.1. How much time do students put into storyboards, as compared to Samba animations? | Diaries |
| 2.2. What do students learn from constructing storyboards, as compared to constructing Samba animations? | Videotape analysis, interviews, participant observation |
| 2.3. To what extent does what students learn from each of the two phases of the assignment resonate with the objectives of the CIS 315 course? | Interviews |
| 2.4. Are both phases of the assignment necessary? Might one or the other be eliminated? | Interviews |

## Students' Work on the Revised Animation Assignment

Twenty-eight groups, each consisting of one, two, or three students, completed animation projects. The animation projects addressed thirteen different algorithms. As illustrated in Figure 62, three particular algorithms or families of algorithms were the focus of over half of

the student projects: Huffman coding (7 student projects); Prim's and Kruskal's minimum spanning tree algorithms (5 student projects); and breadth-first and depth-first search (5 student projects). No other algorithm was the focus of more than two student projects.

Student diaries, coupled with participant observation and interviews, provided insight into the ways in which students went about their animation projects. The following three subsections report my observations, which include the amount of time they spent, the activities they undertook, and their approaches to the assignment.

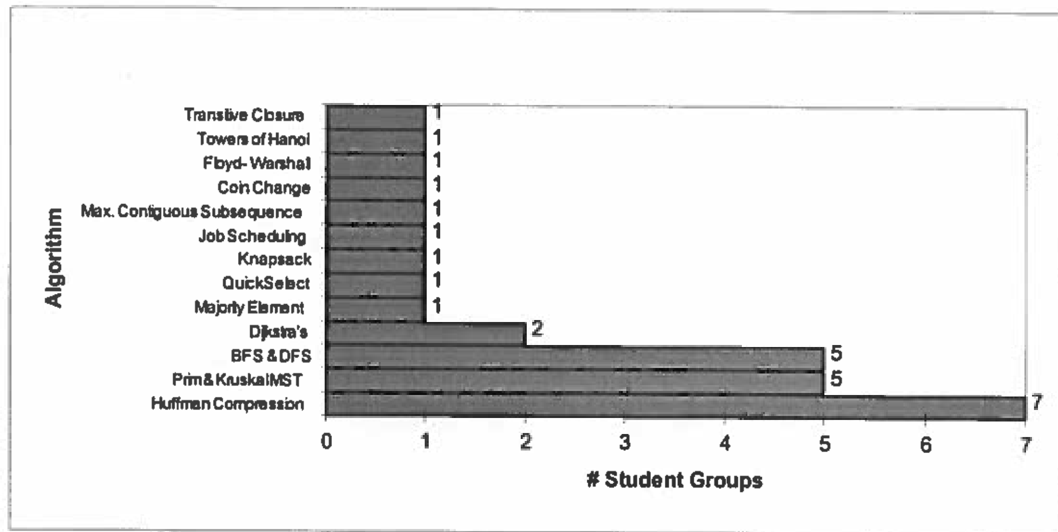



Figure 62. The Animation Projects Chosen By the 28 Student Groups

## Time Spent

The inclusion of diaries in the animation assignment yielded a larger sample of student respondents than did the e-mail surveys of Study I; twenty-seven of the 47 students registered in the course[72] turned in diaries documenting their storyboard-building activities (7), their Samba-building activities (7), or both (13). The level of detail of the diaries varied considerably. While all provided estimates of how much time they spent on the assignment, some of them provided more detail on the nature of the activities they undertook, and on the problems they encountered.

An analysis of the diaries suggests that students spent an average of nearly 40 hours ($n = 20$)[73] on the entire assignment. This total does not include the time they spent in the storyboard and final animation presentation sessions, which are discussed later. As

[72]According to the grade roster provided by the registrar's office.

[73]The sample size was 20 both for the storyboard diaries, and for the final animation diaries, since 13 students provided diaries on both, 7 additional students submitted storyboard diaries, and 7 additional students handed in final animation diaries.

illustrated in Figure 63, just over six of those hours (16%) were spent on preparing their storyboards, while slightly more than 33 hours (84%) were dedicated to the final animations.

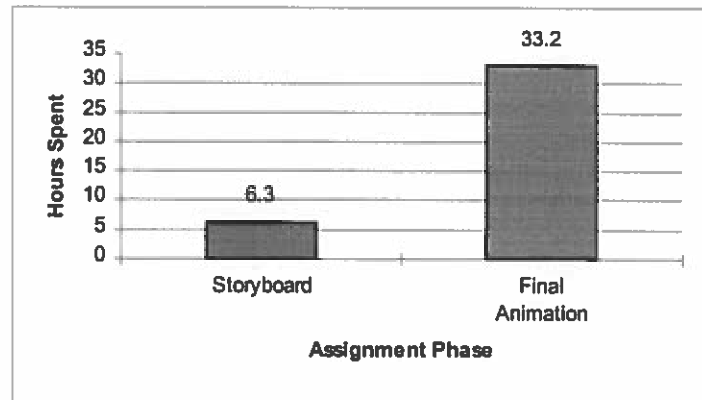



Figure 63. Comparison of Time Students Spent on the Storyboard and Animation Implementation Components of the Assignment

## Animation-Building Activities

Substantial variability in the level of detail that students provided in their diaries prevented me from performing a more detailed quantitative analysis of the time they spent on various activities. I can, however, cite three noteworthy trends in the diaries of those who furnished detailed breakdowns of what they did.

First, for the storyboard phase of the assignment, students spent their time on five main activities: (a) doing group or individual research on algorithms; (b) engaging in group discussions regarding the design of their animations; (c) preparing their storyboard materials; (d) rehearsing their presentations; and (e) actually presenting their storyboards to their professor and peers for discussion. I shall examine the storyboard component of the assignment more closely in below.

Second, 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 (a) writing and modifying algorithms; (b) annotating algorithms with Samba statements[74]; (c) debugging their animations; and (d) tweaking their animations.

Third, a few students spent large amounts of time learning the languages in which they implemented their final animation. While most students chose to use Samba as their final animation implementation language, six student groups opted instead to use Java, which they happened to be learning in a course in which they were concurrently enrolled. Students in at least two of the Java groups complained that they had great difficulties figuring out

---

[74]Clearly, both (a) and (b) apply only to those student group (at least seven, according to the diaries) who elected to use an implemented algorithm to generate their animations.

how to do their animations in Java. On the other hand, six of the Samba groups claimed they spent an average of nine hours learning Samba.

Approaches to the Assignment

The absence of an input generality requirement meant that students in Study II had greater freedom in their choice of animation implementation strategies than did the students in Study I. Indeed, since students were only required to present a single, well-chosen example to illustrate their algorithm, they did not necessarily need to implement the algorithm. 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 (see Figure 64).

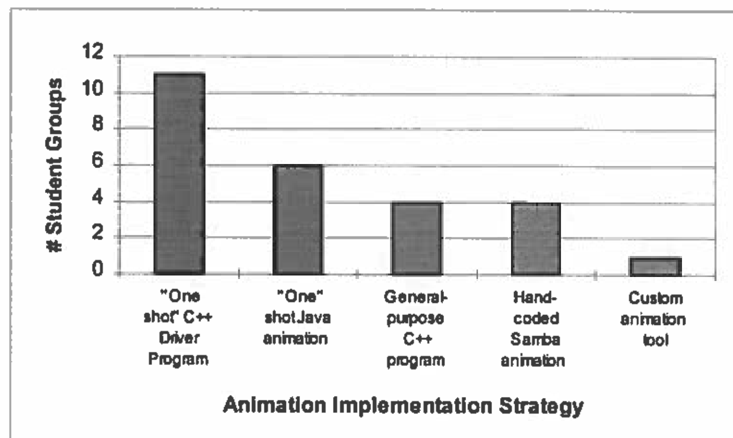



Figure 64. The Number of Student Groups Adopting Five Alternative Animation Implementation Strategies.

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. Students who used Java cited the possibility for greater interactivity (mostly in the form of buttons), and an interest in further exploring Java (which they already were learning for another class), as Java's key selling points. Third, although input generality was not a requirement, four student groups nonetheless chose to implement general-purpose animations. Like the general-purpose animations of Study I, all of these were implemented in the form of a C++ program that accepted general input, and outputted a Samba trace file. 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.

The Presentation Sessions

As was the case for Study I, the presentation sessions of Study II 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 final presentation sessions.

### Storyboard presentations

Heeding the advice of the handout "Guidelines for Developing a Storyboard Presentation" (see Appendix C)[75], 22 of the 24 groups that presented storyboards[76] made use of some sort of pre-prepared visual aid; the other two gave "chalk talks" at the whiteboard. Nearly half of the 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).

Most storyboard presentations lasted between ten and twenty minutes. Students portrayed their animations by presenting illustrations ("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.

Some students included initial screens that introduced the algorithm(s) to be animated. After describing any such introductory screens, students proceeded to provide play-by-play narration of their storyboard. 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.[77]

---

[75]I based the guidelines on my experiences with using the visualization storyboarding technique in prior research (Douglas, Hundhausen, & McKeown, 1995, 1996).

[76]Not all groups that handed in final animations presented storyboards. While the storyboard phase was technically requirement, but a few groups had difficulties scheduling their storyboards and ended up not presenting a storyboard. Those groups, according to Professor Lane, were "on their own," meaning that they proceeded with their animation at their own risk—without knowing whether it would be acceptable.

[77]Note that these observations concur with those of prior research into storyboard presentations (Douglas, Hundhausen, & McKeown, 1995, 1996; Chaabouni, 1996).

Student-professor discussions were abundant and often lively during the storyboard presentations. In the following three subsections, I explore the major themes of those discussions.

Clarifying Questions

In trying to follow storyboard presentations, audience members—especially Professor Lane—frequently broke in with questions. Such questions served to clarify various aspects of storyboard presentations, including

1. the sequence of storyboard events (e.g., "Is the table updated before or after you draw the arrow?");

2. what happens between storyboard snapshots (e.g., "How do you get from that slide to this one?);

3. how, precisely, an animation will unfold (e.g., "How will you actually show that table update?"); and

4. the significance of attributes of storyboard objects (e.g., "Why is that edge colored red?").

A couple of examples from the storyboard presentations illustrate these kinds of clarifying questions. An example of question (3) occurred in Charles's storyboard presentation of the Huffman compression algorithm. For his storyboard, Charles presented just the final snapshot of the Huffman tree. John proceeded to press Charles repeatedly with questions like "What will I be seeing on the screen?"; Charles found himself having to supplement his original storyboard with sketches on the whiteboard. An example of question 4 can be found in Chris's proof of the Prim minimum spanning tree algorithm. Chris's storyboard included side-by-side illustrations of an identical graph. To one graph, Chris applied Prim's algorithm, gradually building a minimum spanning tree. The other graph contained a spanning tree that was assumed to be optimal. At one point in Chris's presentation, John wondered why edges in the graph containing the optimal spanning tree were shaded if they were not, in fact, part of the optimal spanning tree. The following dialog ensued:

C: Yeah, that's a problem that I've got with this. Somehow I want to indicate that this edge right here (points to edge in the optimal solution) was taken in [the Prim solution].

J: The minimum spanning has the edge, or doesn't have the edge?

C: It does not.

In this case, John's question revealed an aspect of Chris's storyboard that he knew would be ambiguous. A discussion followed in which John and Chris considered his options for redesigning the animation. As we shall see in the following subsection, such design discussions were common in storyboard presentations.

Design Discussions

During storyboard presentations, 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. On the other hand, other suggestions were collaborative achievements; they arose out of discussions in which Professor Lane and student presenters considered alternative designs.

As indicated in Table 18, six kinds of suggestions, revolving around six general questions, were offered and elicited during storyboard presentations.

Table 18. The Six Kinds of Questions That Were Offered and Elicited During Storyboard Presentations

| Suggestion Type | Defining Question |
|---|---|
| S1 | What aspects of an algorithm should we illustrate and elide? |
| S2 | How should those aspects of the algorithm be represented? |
| S3 | What are appropriate sample input data? |
| S4 | How should multiple animation views be coordinated? |
| S5 | How can a given storyboard feature be implemented in Samba? |
| S6 | What is the appropriate scope of our project? |

A closer examination of one particular storyboard presentation serves to illustrate all six of these kinds of suggestions. Like several other students, Cara and Rachel chose to compare and contrast two algorithms that solve the same problem. In their case, the problem was one that had been previously covered in class—that of finding a *minimum spanning tree* for a connected, undirected graph (see, e.g., Cormen, Leiserson, & Rivest, 1990, ch. 24).

At the beginning of their storyboard presentation, Cara presented a transparency containing identical, side-by-side graphs; the left one was labeled "Kruskal," while the right one was labeled "Prim." She then noted that a student enrolled in the previous offering of CIS 315 had warned her of the difficulties of running two animations concurrently. Based on that warning, she said that she did not expect that her group's final (Samba) animation would depict both algorithms running simultaneously.

Cara proceeded to focus in on the details of the Kruskal animation. Noting that we should assume that the edges had been sorted by non-decreasing weight, she put up a transparency depicting a single graph that matched those of the previous transparency, except that certain edges were darkened. Pointing to the darkened edge between vertex *h* and *c*, she said that the edge was the first one picked by the algorithm. Pointing to the edge between vertex *d* and *e*, she stated, "If we look at the queue, this will be the next one, and that goes on."

After answering a couple of Professor Lane's clarifying questions, Cara turned the presentation over to Rachel. Rachel began by restating what Cara had just explained: "I was thinking, at each step, if an edge is taken, it is highlighted, and it will be explained in this window (points to area below graph) why it was picked. Putting up a transparency depicting the same graph as before, but with certain vertices darkened, she proceeded to explain the animation of Prim's algorithm: "It would start from any arbitrary vertex (points to vertex *a*), and then it would look at. . .other vertices (points to vertex *b*) and check if [they] have an edge that connects to any vertex in the tree (points to edge between vertex *a* and *b*). And then it would. . .pick the one that has least weight, and connect it to the tree."

A brief silence followed, and John broke in with the first of his suggestions, which addressed the synchronization of the Kruskal and Prim animations (suggestion type S4). In the following monologue, John notes that it would be not only possible, but also instructive, to show the Kruskal and Prim animations unfolding simultaneously:

> One thing you said along the way is that you don't think it would be possible to run both of them at the same time. But you can show some sort of sample graph. . .One of the things I mentioned in class [is that] once you have generated the Samba script, it may be possible—given that the whole thing isn't very big—to interleave the two. . . .Especially the step that picks the next edge—when both pick their edges, you can actually be seeing what's happening at that point.

This suggestion naturally led to an example of suggestion type S5. In order to give Cara and Rachel some advice on how they might implement side-by-side, concurrent animations in Samba (which, at the beginning of their presentation, Cara had said they wanted to avoid), John called on me as the resident Samba expert. I offered the following as a possible strategy:

> You're just doing a couple of samples, so it becomes tractable. . .to do some file merging after the fact; you can merge the scripts. . . .As long as you documented your Samba well enough, you should be able to figure out how to synchronize [your scripts] pretty simply. . .In Samba, you express concurrency by putting the things in brackets that you want to happen at the same time. . . .So then you can interleave in brackets the things that are gonna happen together.

Exemplifying a suggestion of type S3, John's comments then turned to the group's choice of sample graphs. As he saw it, the sample graph they had chosen may not allow them to show the two kinds of comparisons between Kruskal's and Prim's algorithms that one typically wants to make:

> If you pick a graph with all different edge weights—you didn't do that here—. . .then the minimum spanning tree is unique, even though Kruskal and Prim are going to build them differently. . . . If you want to illustrate that they don't necessarily wind up with the same thing, . . .then you want to have lots of coincidences of edges. Make all of the edge weights 1, 2, 3, 4,. . .and then there are going to be a lot of different spanning trees.

John then scrutinized what, exactly, Rachel and Cara's storyboard was illustrating. In the following exchange, John first recalls Cara's presentation of the Kruskal animation. Then, in an excellent example of suggestion type S1, he notes that it would be instructive to show the list of candidate edges being considered for the minimum spanning tree:

> J: You mentioned that you were going to say something in the window about when you chose the minimum.
>
> R: Right.
>
> J: And then you sorted the edges, but we're not seeing them there. . . .It doesn't really matter, at this point, how you sorted them;. . .for a small graph you could have just done it by hand. [Now], you could actually show [the edges] in another window.

Notice that, while John saw the selection of candidate edges as significant, he did not believe the way in which those edges are sorted to be significant. Thus, he recommended not only what to show, but what *not* to show.

Taking his suggestion even further, John went on to recommend some ways in which the selection of candidate edges might be illustrated—an example of suggestion type S2:

> And you sort of highlight [the candidate edge], or cross it out, when you're done. . . .As you go through them, you have to consider every edge. And you can. . .erase if it's not used, or leave it there if it is used,. . .or color them in different colors.

Rachel and Cara followed up John's suggestion with an idea of their own. Instead of just showing the selection of candidate edges, they thought it would instructive to show the reason why a particular edge was *not selected*:

R: I was thinking of highlighting the edge [being considered]. If it is not picked because it is. . .

C: . . .creates a cycle

R: Yeah, then I would explain [why] in a window underneath.

Noticeably excited, John answered by praising their suggestion and expanding on it:

> Oh, that would be nice. It would be nice if you can highlight the cycle, so you can actually show, here's a cycle. You can use yet another color for that. Temporarily you show this whole cycle flashing. Sure, that would show this thing working.

This kind of exchange, in which Professor Lane and students developed design ideas collaboratively, can be found in several of the storyboard presentation sessions. [How many?]

At the beginning of the presentation, Cara mentioned that she and Rachel were planning to illustrate the two algorithms' efficiencies. At the end of their presentation, however, Rachel conceded that they were not sure how, or whether, they could do that. In the following exchange, Rachel's question gives rise to a suggestion of type S6. In particular, while he points out that there are interesting possibilities for illustrations of efficiency, John recommends that they limit the scope of the animation to how the two algorithms work:

R: I was wondering if we have to show the time of each algorithm.

J: What you're illustrating here so far is how the algorithm works. There's a lot in terms of timing. You can get into how you sort the edges in the first place, . . .how you know whether it's a cycle when you're picking it. . . . You have to get into some fancy data structures in order to get some efficient timing. . . .There's a whole chapter—I think it's chapter 10—on how to do this kind of thing, which is whole thing to illustrate all by itself. I think the timing. . .is probably not what you want to illustrate here. I think you should concentrate on illustrating the algorithms themselves.

The storyboard presentation thus concluded. As I have illustrated, John, Cara, and Rachel considered six distinct kinds of design suggestions along the way.

Tailoring Scenarios to Algorithms, and Algorithms to Scenarios

Cara and Rachel's storyboard was based heavily on the established representations and symbolism used in the course textbook (see Cormen, Leiserson, & Rivest, 1990, pp. 506–508). Specifically, they represented their graphs as collections of labeled circles with lines drawn between them, and they darkened vertices and edges to indicate their inclusion in the minimum spanning tree being grown. In fact, sixteen of the 24 storyboard presentations represented algorithms as Cara and Rachel did—that is, as purely geometric objects in a process of transformation. While a few of those sixteen pioneered novel geometric representations, most were based, at least loosely, on established representations found in the course textbook.

On the other hand, eight of the 24 storyboard presentations depicted algorithms against the backdrop of a *scenario*, in which real or fictitious human beings were engaged in some problem-solving venture. Derived either from the real world or from a fantasy, such scenarios were carefully developed so that their internal logic and structure highlighted (and masked) various aspects of the algorithms they portrayed. As it turned out, storyboard presentation participants took great pleasure in refining and further developing the internal logic and structure of these scenarios. This was especially true if the scenarios were creative or innovative, as were the five summarized in Table 19.

During the course of the scenario-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. To illustrate such discussions, let me describe an example drawn from the storyboards summarized in Table 19.

For his storyboard presentation, Gerald devised the scenario of "Knuth's Ark": The world floods again, and Donald Knuth[78] 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?

Table 19. Five Storyboard Stories that Stood Out for Their Creativity and Innovation.

| Algorithm(s) | Scenario 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 $n^{th}$ heaviest nugget in a given batch is to be selected. How can the Select Mining Corporation select it most efficiently? |
| Longest common subsequence | Knuth's Ark | The world is flooded once again, and Knuth's Ark, which contains a pen of wild animals, lands on an island with a pen of wild animals of its own. Knuth's Ark has limited space, and Knuth must select only those animals on the island that have mates on the Ark. Which ones should he select? |
| 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? |

[78]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)

After introducing this story, Gerald explained that Knuth's strategy would be to employ an algorithm for solving the longest common subsequence problem.[79] However, John immediately detected a discrepancy between that problem and Gerald's scenario. As he put it, "[Y]ou have to justify. . .why you're doing this [loading of the Ark] in sequence, as opposed to finding a subset." John went on to share his idea for reconciling the story with the longest common subsequence problem:

> So these guys are already loaded on the ark, and they're in their stalls, and they're all pushed to the end. It's like on a ferry—the first cars in, you can't move them out.. . . Either they come in sequence, or they're out of luck; otherwise you have a lion next to a lamb.

In the dialog that followed, John emphasized the importance of underscoring that the animals not only are in stalls, and also must maintain their order:

J: OK, make sure you get a point like that across, 'cause otherwise you're solving the biggest common subset, which is another problem.

G: Right, that's true.

J: Make sure that they're in stalls, and they can't come out. And the [animals] have to be fed onto the boat, like on a ferry.

G: That's definitely important.

J: Otherwise you're solving the wrong problem.

John's recommendation that animals be confined to linear stalls gave rise to a dilemma: It seemed unlikely that Knuth would find the animals on the island confined to such stalls; quite simply, the scenario of Knuth's Ark seemed incompatible with the longest common subsequence problem.

In the lively discussions that followed, John, Gerald, and I attempted to resolve this dilemma using the two alternative strategies used in other scenario-based storyboard sessions: (1) tailoring the algorithm to the scenario, and tailoring the scenario to the algorithm.

*Tailoring the algorithm to the scenario.* First, we considered whether Gerald might illustrate an algorithm that better matched the "Knuth's Ark" scenario. In fact, Gerald himself conceded that the scenario was inspired by a different problem altogether—the greatest common *subset* problem.[80] This was revealed after he presented the first two snapshots of his storyboard. John detected a problem, and jumped in:

J: You're missing something.

G: I think you're right. I think what I'm solving is the greatest subset problem.

---

[79]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 (Cormen, Leiserson, & Rivest, 1990, pp. 314–319).

[80]Given two unordered *sets* of objects (possibly containing duplicates), what is the largest subset that the two sets have in common?

Plainly, the match between Gerald's scenario and the greatest common subset problem had a lot going for it. For one, the animals on Knuth's Ark, as well as those on the island, seemed clear instances of *unordered* sets. Indeed, in the real world, they would be able to mingle among themselves freely; they would not be confined to linear stalls. Furthermore, when breeding season came around, it seemed obvious that each animal would be smart enough to "sniff out" its mate, assuming that such a mate existed. In other words, no ordering of the animals—using stalls, or some other mechanism—would seem to be necessary.

Given these parallels between Gerald's scenario and the greatest common subset problem, we wondered whether Gerald should pursue an animation of that problem instead. Since the problem had not been covered in class, a discussion ensued in which we considered what kind of algorithm one could use to solve it. The first issue was whether or not one is allowed to sort the elements. For, as John pointed out,

> If these are elements that I can sort, then you can sort both them, and start matching [them] up, and you can essentially do some sort of merging. . . .It's a completely different algorithm. This is under the assumption that sorting makes sense for the two different [sets].

With respect to Knuth's Ark, John, Gerald, and I ultimately decided that sorting animals may not make sense:

E: It's too hard to sort wild animals! It was hard enough to get them in a line!

J: Yeah.

G: Unless you had them each in individual cages.

E: That's too much work.

G: Yeah, that's a lot of work.

J: Yeah, that's not plausible.

Given that sorting did not make sense in the Knuth's Ark scenario, the second issue to resolve was whether or not dynamic programming could be used. In light of CIS 315's emphasis on dynamic programming, Gerald had every intention of illustrating dynamic programming. If it turned out that some other programming technique could be used to solve the largest common subset problem as efficiently as, or more efficiently than, dynamic programming, then the problem would probably not be an appropriate one for animation. After giving it some thought, John decided that dynamic programming could, indeed, be used to solve the problem, but that a naïve algorithm could be used to solve the problem just as efficiently:

> I can give you an n-squared algorithm that's not dynamic programming. . . .You have to make a pass through the other set [of animals on the island] for each [animal on Knuth's Ark].

John's discovery led us to conclude that the largest common subset problem was probably not a good one to illustrate:

J: The thing is, it's hard to justify dynamic programming when you can get an n-squared algorithm that isn't.

G: Yeah.

J: So that's why the common subsequence is a more interesting problem from that point of view—algorithmically.

G: 'Cause that's the best way you can do it.

J: Yeah.

*Tailoring the scenario to the algorithm.* Once it became clear that animating the greatest common *subset* problem might not be appropriate, we turned our attention to modifying the "Knuth's Ark" scenario so that it became a plausible backdrop for the longest common subsequence problem. In the following exchange, we tried to make the best of John's original idea that the animals be confined to pens:

E: OK, you can say that the island has a queue that's fixed, and you can't shuffle things around.

G: Yeah, right, same thing with the boat.

J: You're loading all of these animals, and you have them in this pen, and you can't do anything about it.

G: Yeah, the people on the island have also done the same thing.

E: OK.

J: Well, that's as realistic as anything else you've said.

(*Laughter.*)

At the end of his presentation, Gerald asked for advice regarding how best to lay his animation out. His request stimulated the following brainstorming session, in which we further developed the scenario to accommodate the longest common subsequence problem:

E: You might want to rotate the [island] pens so that they're in the same direction as the ark.

J: Yeah, you could have a plank. You have to load [the animals] onto the ferry some place. You could have a dock coming out.

E: (*Points to shoreline tree in Gerald's storyboard.*) Yeah, get that tree out of the way!

J: OK, and I mean, the animals that don't make it, just shove them off [the dock]!

(*Raging laughter.*)

E: And splash!

(*Laughter.*)

In sum, scenario-based storyboard presentations generated discussions in which participants took great pleasure in reconciling algorithms with scenarios, and scenarios with algorithms.

## Final Animation Presentations

Twenty-nine groups, not all of which had presented storyboards, presented final animations. Because he was out of the country for part of the term, John was able to attend only 8 of these presentations; the course teaching assistant (Tom) and I moderated the 21 presentations that John missed.

On the one hand, the final presentations were similar to the storyboard presentations in two key respects. First, their content was generally predictable, given the corresponding storyboard presentations upon which they were based. Generally speaking, students took

into consideration the comments and suggestions they had received on their storyboards; their modified storyboards served as specifications for implementing their final animations. One notable exception to this was Curt and Jeff's animation of the knapsack problem. Not only did Curt and Jeff heed John's advice (an example of S6; see above) to compare the fractional knapsack solution with the 0-1 knapsack solution, but they also concocted the humorous "Beavis and Butthead" scenario summarized in Table 20. Given their original storyboard, which included random thieves, one would not have predicted that they would have developed such an elaborate story for their final animation.[81]

Table 20. Curt and Jeff's Final Animation Scenario

| Algorithms | Scenario Title | Synopsis |
|---|---|---|
| Fractional vs. 0-1 Knapsack | "Beavis and Butthead" | Beavis and Butthead are making mischief again. Consistent with their personalities, they use a greedy strategy to fill their sack with the combination of gold, silver, emerald, and coal that has the greatest value. But when they try to use the same greedy strategy to rip off discrete items (e.g., computers, printers) in the Computer Science Department, it fails. In a flash of Beavis insight, they adopt a dynamic programming approach, and proceed with their robbery. |

Second, although the level of detail of storyboards and final animations differed markedly, what students ended up showing—precisely those portions of the animation that they deemed interesting or unusual—ended up being the same. In other words, although they often went to great pains (over 30 hours' worth, as we saw above) to implement their animations in full detail, when it came down to presenting their animations, they tended to fast-forward through much of that detail. Instead, they tended to focus on the same portions of their animations that their storyboards illustrated exclusively—the interesting sequences, in which something noteworthy or unusual occurred.

On the other hand, aside from being generally shorter in duration than the storyboard presentations, the final presentations differed from the storyboard presentations in two key respects. First, there was noticeably less discussion overall. Instead, 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 addressed topics that were different from those of the storyboard sessions. Whereas storyboard presentations were replete with both design discussions and discussions regarding scenarios, final presentation sessions were generally devoid of them. Notable exceptions to this were those final animation presentations that did not benefit from a prior storyboard presentation. In some such cases, students simply had not presented a storyboard; in others, students had presented a storyboard, but ended up significantly altering the scope and focus of their final animation in response to a suggestion from John (suggestion type S6). In either case, the final

[81]although, at one point during the storyboard, John did comment that "you can even write a story [about] all of this."

presentation sessions served as *de facto* storyboard presentations; the same kinds of clarifying questions were asked, [82] and the same kinds of design discussions took place.

Moreover, whereas storyboard presentation discussions occasionally addressed implementation *considerations* (suggestion type S5; see above), final animation discussions frequently revolved around implementation *details*—how something was implemented, difficulties encountered during implementation, and the like. Occasionally, such discussions rivaled the "focus group discussions" of Study I (see Appendix A), as 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 final presentation sessions; indeed, in addition to sharing their animations, some students simply felt inspired to share how they had implemented them.

### Was the Revised Animation Assignment Format an Improvement?

Having presented the main observations of Study II, I am now in a position to address the study's research questions. In this section, I consider the first, core group of research questions, while the next section considers the second, exploratory set of research questions.

#### Did a Single Animation Assignment Work out Better?

Whereas Study I included three animation assignments worth 21% of the total grade, Study II included a single animation assignment that counted for just 16%. In Study II, students did not start thinking about animation until around the halfway point of the term, and they finished their final animations a week in advance of the final exam. As a result, students spent less overall time on their animation assignments; further, according to John, they did not take time away from their final exam preparations to get their animations done. In my exit interview with John, I asked him to assess the new assignment format:

**E:** Do you think that the format of assignments this past spring term was an improvement over winter term—just to have one assignment?

**J:** Yeah, that part was better. They were a little bit better directed in what they were going to be doing. In the winter term, some of them did better projects for the five points than they did for the final one. They put in all this time and effort. No, I think this worked out somewhat better. It would be interesting to see if we could present them with better graphical tools.

**E:** So they can do more assignments?

**J:** Right, so they could spend a little bit more time thinking about the algorithm than in making the pictures.

Thus, John's only concern was that the tools that students used to construct their animations (Samba and Java) required them to spent too much time on creating graphics, and not enough time on the algorithms. Given that the tools students used for their final animations did not change from Study I to Study II, this was to be expected.

---

[82]There was one other case in which clarifying questions were frequently asked: In those presentations in which Tom and I filled in for John, Tom frequently asked clarifying questions. This is not surprising, given that he had not attended the storyboard presentations.

Was the Absence of an Input Generality Requirement an Improvement?

Eliminating the input generality requirement appeared to be a good decision, both from the perspective of students, and from the perspective of Professor Lane. Like students in Study I, students in Study II complained of the difficulties of graphics layout. However, unlike the students in Study I, the students in Study II were able to address the problem by heeding the revised assignment's recommendation to focus on just one example, rather than implementing a general-purpose animation. Consider, for example, the following conversation, in which Laura and Gayle discussed the motivation behind their decision to hand-code their animation of the Huffman compression algorithm in Samba:

> L: It seemed like the difficulty with the animation was primarily positioning stuff. . . .And then it became really clear that doing things like getting the tree to draw, by putting the Samba code into C code, was just the way to torture yourself. It just didn't seem like it was going to be workable—possible, but not the smart way to do it.
>
> E: Especially since you just had to get this running for one example.
>
> L: Right.
>
> G: Yeah, I think that's the major thing. If you had to do the general case, like you said they had to do last term, it would be a different situation. But we had exactly one example that we wanted to do, and we knew exactly where we wanted everything laid out. Making it work in the C code would be for a generalization. We just wanted a drawing, making this be Huffman. And we could do a better job on the one example, just laying it out exactly in Samba.

As Gayle pointed out, being able to focus on just one example enabled them to do a better job on their animation, And, as Professor Lane noted after their storyboard presentation, it was precisely because Laura and Gayle's animation was not general purpose that it succeeded:

> And the way [Laura and Gayle] are illustrating the Huffman coding I think is better than what I've seen in other animations. . . .So they're actually taking something, by not making this an all-purpose program, they're actually.. .illustrat[ing] something about how this compression is working. You can actually see this happening with the steps. . . .They seem to be focusing more or less on the important parts of the algorithm, without worrying about the implementation. . . . I think that is potentially a very good animation showing how an algorithm works.

Laura and Gayle's experience seemed to be common among my student informants. Freed from the tremendous burden of implementing general-purpose layout algorithms (see Appendix A for the difficulties students in Study I encountered), they were able to concentrate on the important details of their algorithm.

There was, however, one notable case in which a group of students became so wrapped up in a single example that they seemed to overlook the important aspects of an algorithm. Dave, Josh, and Gabriel developed an animation that first illustrated Prim's minimum spanning tree (MST) algorithm, and then proved its correctness. In an interview with them, I asked about their approach to the assignment. Like Laura and Gayle, they had elected to hard-code a single example of an execution of Prim's algorithm. And like Laura and Gayle, they felt that focusing on a single example had helped them:

> D: [Our animation] would only work for one example.
>
> E: Oh, right. So you don't have. . .
>
> G: Prim's algorithm running underneath this.

E: Right. . . .

D: So we haven't really been playing with Prim's algorithm.

E: It's just hard-coded.

D: Hard-coding this one example of Prim's algorithm. I'm sure we know that example backwards and forwards. But that means we know the algorithm.

The idea of their proof was to compare the MST constructed by Prim's algorithm with a MST that is known to be optimal. At points where the Prim algorithm had selected an edge that was not in the optimal MST, they added that edge to the optimal MST. When they were through adding edges to the optimal, they went through the optimal MST and got rid of all the cycles by eliminating all edges that were not in the Prim MST. Since the total weight of the optimal MST did not increase as a result, they conlcuded that the Prim MST must be optimal.

However, in reviewing their final animation, John noticed a crucial flaw in their logic: namely, if one waits until the end of the proof to eliminate all of the cycles, "when you're all done, you don't know which cycles you've formed, and which cycles were formed because other cycles overlapped." Moreover, in John's opinion, Gabriel, Dave, and Josh would have recognized their flaw if they had not been so steeped in a single example graph. In fact, he likened their case to the case in which Fred lost the forest for the trees in Study I (see Appendix A):

> They didn't stop to think about what this would look like in a more complicated graph. . . .I think that this was just a higher-level analog of what happened last term with the kid who was drawing the forest and forgot what the algorithm was all about. . . .They wanted to [show a proof] because I told them I'd really be impressed by something which showed a proof. They tried to show a proof, but they didn't really understand what the proof was all about.

That the same kind of "single-example myopia" occurred even when the input generality requirement was in place suggests that the absence of the input generality requirement may not have been the root cause. Rather, it may be that some students simply have a tendency to focus too intently on a single case. Whatever the case, the absence of an input generality requirement appeared to be an improvement over Study I; students were generally able to construct animations that, at least in Professor Lane's opinion, better illustrated the algorithms they depicted.

### Assessing the Storyboard Phase of the Assignment

Recall that Professor Lane and I included the storyboard phase in order to foster increased professor-student interaction about students' animations. We wanted to foster such communication for two reasons. First, we hoped that it would lead to improved learning experiences, in which students focused on the appropriate aspects of the algorithms they were animating. Second, we hoped that it would make it easier for John to grade the assignments, since the storyboard phase could be used to establish, on a case-by-case basis, a "contract" of what was expected.

The storyboard presentations provided students not only motivation to think through an animation carefully, but also an opportunity to confirm that such thinking was on track—"in the spirit of the assignment," as John often put it. Observations presented in above suggest that storyboard presentations were replete with professor-student interaction in which

design improvements were offered, discussed, and collaboratively refined. Insofar as such communication served to put students on track by reinforcing what they should be focusing on, the storyboard presentations can be seen to have fulfilled the first of our goals. As John mentioned in our exit interview, this was especially true in cases in which students' storyboards indicated that they were *not* on track:

> We saw a few storyboards where the students didn't know what they were talking about. It certainly became clear, and hopefully they did better [on their final animations] in those instances.

Those students whose storyboards indicated that they were on track certainly benefited as well. Indeed, as John noted, they invariably went on to develop high-quality animations: "All of the storyboards that I remember being good. . .turned out to be good animations." An excellent example of a good-storyboard-turned-good-animation can be seen in Laura and Gayle's Huffman animation, on which I touched in the previous section. With respect to their project, John commented that "I saw everything in the storyboard already. . . .[The final animation] was exactly what I expected, and worked out very nicely on the screen."

Whether the storyboard phase of the assignment made grading any easier for John was less clear. As was the case in Study I, John was concerned in Study II both with the difficulty of grading the assignments, and with the amount of time he was spending on grading. Even if the design discussions in the storyboard presentations succeeded in establishing implicit contracts of expectations between John and student groups, the fact remained that John still had to grade the assignments subjectively. As he confessed, that was something with which he was simply not comfortable: "I am always very weary of having to grade subjectively in part, and it's very difficult to write down the criteria." Furthermore, he still had to review the final animations, again relying on both my videotaped footage and the actual animation executables that students deposited in the course directory. Once again, such review was a time-intensive process:

> The problem about the grading: It takes a long time to go through some of them; you have to review them several times to really see whether they really know about it. . . .[T]he ones that really looked interesting and most flashy and most colorful and most animated don't necessarily show the most about the algorithms. So, that was really difficult.

However, John did acknowledge that the eight presentations (out of 29 total) that he actually attended were easier to grade, since he did not have to review them as much:

> So, if I actually saw them, I didn't have to really review the animation. Halfway through [them], I knew exactly what the grade was going to be.

In sum, while the addition of the storyboard phase may have improved students' grades, it did not necessarily make grading any easier for John. On the other hand, just being at the presentations appeared to make grading easier.

<u>Comparing the Storyboard Phase to the Final Animation Phase</u>

The second set of research questions proposed by Study II were interested in comparing the storyboard phase with the final animation phase of the assignment. The first of those asked how students' time and activities varied between the storyboard and final animation phases of the assignment. As we have seen, the observations earlier provide definitive answers. Students spent roughly five times as much time on the final animation as they did on the storyboard phase of the assignment (33.2 versus 6.3 hours; see Figure 63). Moreover,

students' activities in each of those phases differed fundamentally: Whereas they spent most of the storyboard phase on research, group discussion, and preparation of visual aids, they spent most of the final animation phase on implementation activities, including coding, debugging, and tweaking.

In light of these substantial differences, it is important first to investigate any corresponding differences in what students learned from each phase, and then to consider the relevance of what they learned vis-à-vis Professor Lane's teaching objectives for CIS 315. At that point, one should be in a position to make statements regarding the effectiveness of each phase, and hence how to improve the assignment. I turn to each of these questions in the following three subsections.

### Differences in What Students Learned

Early in my Study II fieldwork, I sensed that what students were doing for their final animations differed significantly from what they were doing for their storyboards. Accordingly, in my semi-structured interviews with student groups, I made a point of exploring students' own perceptions of how the two phases of the assignment differed. In those interviews, students' consistently made the same three points:

They knew the algorithm by the time they had finished the storyboard presentation.

The process of constructing the storyboard was helpful in cultivating and refining that understanding.

Their understanding of certain details of the algorithm occasionally changed through implementing the final animation.

Vignettes of my interview with Laura and Gayle serve to illustrate these points. When asked whether they had learned the Huffman algorithm in the storyboard or final animation phase of the assignments, Laura and Gayle concurred that they had learned the algorithm prior to implementing their final animation:

> E: So, do you think that before you started the Samba part of it your grasp of Huffman was reasonable? Or do you think that doing the Samba helped more with Huffman? Was it all kind of pre-determined when you did the storyboard?
>
> L: Yeah.
>
> G: We knew Huffman.
>
> L: [For our final animation,] we're doing just what we said we would do in the storyboard. . . .
>
> E: So it's become just implementing a spec., kind of?
>
> L: That's right.

Furthermore, the process of developing a storyboard aided in their understanding. As Laura put it,

> [Building a storyboard] is like writing the pseudocode. We got it all sorted out, exactly what we were going to do and when, and how we were going to coordinate these pieces together, because we had to have a coherent presentation for the storyboard thing.

However, when it came down to implementing the details of their storyboard, Gayle confessed that her understanding of one particular detail of the algorithm changed:

> The only thing that I've changed my mind a little bit . . .is. . .that we actually are only going to go through the text twice. And [for the storyboard phase] I thought that Huffman went through it three times. But building the tree, you don't actually have to go through every bit of the text to do it, because you already have the frequencies. So [our final animation]. . .show[s] that you only go through the text twice in order to first, find the frequencies, and then encode the path.

According to Gayle, her change in thinking was brought about by the need to implement every single letter in the file as a Samba object—something that she did not do for the storyboard:

> [W]e have to make one [Samba] object for every text letter both times that we go through the code, and think then how many times you actually have to go through every letter in the file to do this process.

In sum, it appeared that students had a pretty clear idea of the value of each phase of the assignment. On the one hand, by studying candidate algorithms, and by ultimately coming up with high-level designs of animations for illustrating those algorithms, they believed that they actually learned the *algorithms* in the storyboard phase. On the other hand, in the final animation phase, they turned their attention primarily to figuring out *how to implement* their animations; they learned both a particular programming framework (Samba or Java), and how to manifest their storyboard designs in that framework. On occasion, tending to all of the details of the animation implementation helped them to refine their understanding of certain details of the algorithm. By and large, however, the learning of the algorithm took place during the storyboard phase.

### Relevance of What Students Learned

Clearly, the effectiveness of each phase of the animation assignment hinges the extent to which it is *relevant* to the CIS 315 course; learning activities that do not address the course's objectives simply cannot be effective. Given what students believe to have learned during each phase of the assignment, the next research question to be considered is, To what extent was each phase relevant to CIS 315? To answer that question, let me first revisit the goals of CIS 315.

As discussed in Chapter 3, CIS 315 aimed to familiarize students with the design and analysis of efficient algorithmic problem-solving techniques. The three central questions of the course were

1. Given a problem, what is an efficient way to solve it?
2. How efficient is the solution, in terms of Big-O notation?
3. Why is the solution correct?

The course homework and exams, which counted for between 79 (Study I) and 84 (Study II) percent of students' total grade, consisted almost exclusively of problems for which students had to answer those questions. As Professor Lane stated in the course syllabus (see Appendix C),

> Most of the homework [and exam] problems will require the description of algorithms for specified problems. Pseudocode may be useful for such descriptions but is nether necessary nor sufficient. That is, you should explain your algorithm, *justifying both the correctness and timing.*

In both Study I and Study II, John dedicated part of an early lecture to discussing practical tips for how to succeed in CIS 315—that is, how to become adept at answering the three above questions in a satisfactory manner. Above all, John encouraged students to become *self-reflective.* Instead of passively reading the textbook's solution to a given problem, students were asked to consider how they would solve the problem themselves. Further, in order to devise "reasonable" and "convincing" explanations of an algorithm's efficiency and correctness, students were advised to work in study groups. For, as John put it in one lecture, "if you can convince each other, you'll probably do well on the test."

In the mathematics community, it has often been said that "proof is a social interaction." Likewise, John's recommendation that students work in study groups emphasizes that the kinds algorithm design and analysis undertaken in CIS 315 are fundamentally *social.* In John's words, a "reasonable" explanation or justification will "convince someone at your [the students'] level of understanding." Thus, the form of a "reasonable" explanation or justification intimately depends on, and is informed by, one's audience. Without an audience to interact with and to receive feedback from, it would appear difficult, at best, to formulate such "reasonable" explanations and justifications.

Given the social nature of algorithm design and analysis, it seems reasonable to assume that learning activities with the highest relevance to CIS 315 would set up situations in which students talk with each other about algorithmic behavior, efficiency, and correctness. In other words, they should be given opportunities to

1. explain to each other how an algorithm works;
2. justify to each other what an algorithm's efficiency is; and
3. convince each other that an algorithm is correct.

According to this definition of relevance, the storyboard phase of the animation assignment was clearly more relevant than the final animation phase. As we have seen, in the final animation phase, students were primarily concerned with the details of *implementing* an animation. On the other hand, in the storyboard phase, students not only considered, but also discussed with each other and with their professor, one or more of the three central questions of the course listed above. In other words, whereas the final animation phase afforded students the opportunity to delve into implementation details, the storyboard phase of the assignment afforded students the opportunity to engage in one or more of the three forms of social interaction that are highly relevant to CIS 315.

## Are Both Phases Necessary?

If it is true that the storyboard phase of the assignment had greater relevance to the CIS 315 course than the final animation phase, then what is the implication for improving the assignment? Should the final animation phase simply be discarded, or do both phases have important contributions to make?

In my exit interview with John, we discussed that question. When asked about the possibility of eliminating the final animation phase of the assignment, John readily defended the value of the final implementation phase:

> Well, I think there would have been a problem with their appreciation of it, if they didn't go ahead with the implementation—if they just came up with a storyboard of how they were to animate it, if they ever were to animate it. I think it would have kind of left a hole with them. They were doing the storyboard because they were planning out an animation. . . . [T]hose that came up with nice projects really seemed to enjoy the implementation that they had.

We see, then, that in John's view, the storyboard phase alone, while perhaps highly relevant to the course, would be incomplete without the final animation phase—at least from the students' perspective. John's assumption was that, as computer science students, the students in CIS 315 naturally enjoyed implementing their animations on the computer. After all, these students presumably decided to become computer science majors because they enjoy programming and working with computers. Carrying John's reasoning further, I commented in my final interview with John that the final implementation phase "was kind of a driving force for [students]"—a reward, of sorts, for completing the storyboard phase. In sum, while the final animation phase may not have substantially contributed to students' "learning" the algorithm, it may have had important motivational value. Future fieldwork could pursue this issue further by eliciting students' views on the role of both assignment phases.

## Discussion

For Study II, Professor Lane and I made three significant changes to the animation assignments of Study I. First, instead of giving three animation assignments, we only gave one. Second, we did not require that students' animations work on general input. And third, we integrated a novel learning exercise—the storyboard phase—into the animation assignment.

Study II fieldwork suggested that these changes to the assignment effected the desired results. By cutting the number of assignments down to one, we saw students spent less time on algorithm animation overall. By relaxing the input generality requirement, we found that students spent less time on the implementation issues surrounding the "general purpose layout problem." Instead, students dedicated their time to carefully choosing input data, and to getting their animations to look good for just a few examples. Thanks to the storyboard phase, students interacted and collaborated with Professor Lane more extensively than in Study I, leading to improved animations and higher grades. In short, the revised assignment format enabled students to spend more time on animation activities that were particularly relevant to the course: namely, thinking about, discussing and designing visual representations of the behavior, efficiency, and correctness of algorithms.

After having tried algorithm animation assignments for two academic quarters, Professor Lane's impressions of algorithm animation remained generally positive. When asked whether he was interested in giving algorithm animation assignments in future offerings of CIS 315, he commented, "Oh yeah, especially since now we have a better idea yet of how to do this, and especially when you see some of the really good ones that are worth using for teaching the course next time around."

# APPENDIX C

# ETHNOGRAPHIC STUDY MATERIALS

## CIS 315 Syllabus for Study I[83]

| | |
|---|---|
| **Class meetings** | Mon, Wed, Fri, 10:00-10:50 111 Foo |
| **Instructor** | John Lane, 100 Deschutes<br>Office hours: to be announced |
| **Teaching Assistant** | Tom Maly, 200 Deschutes<br>Office hours: to be announced |
| **Teaching Assistant for algorithm animation** | Chris Hundhausen, 231 Deschutes |
| **Prerequisites** | CIS 313, Math 233 |
| **Text** | Cormen, Leiserson, Rivest<br>*Introduction to Algorithms* |
| **Newsgroup** | `uo.classes.cis.315`<br>for assignments, announcements, hints, etc. |
| **Assignments** | Homework problems assigned regularly.<br>Three algorithm animation assignments<br>Optional programming assignment as alternative to final exam |

**Grading**

| | |
|---|---|
| Homework | 19% |
| Algorithm animation | |
| First project | 1% |
| Second project | 5% |
| Third project | 15% |
| First midterm | 20% |
| Second midterm | 20% |
| Final | 20% |
| Extra credit problems | variable credit |

**Important dates**

| | |
|---|---|
| First animation due | Wed, Jan 22 |
| First midterm | Wed, Jan 29 |
| Second animation due | Wed, Feb 12 |
| Second midterm | Fri, Feb 28 |
| Third animation due | Mon, Mar 10 |
| Final exam | Mon, Mar 17 |

---

[83]Certain items in this syllabus have been changed or omitted in order to preserve anonymity.

**Homework**

Most of the homework problems will require the description of algorithms for specified problems. Pseudocode may be useful for such descriptions but is neither necessary nor sufficient. That is, you should explain your algorithm, *justifying both the correctness and the timing.*

The homework assignments will carry equal weight in determining your final homework grade. In computing your final homework grade, the lowest assignment-grade will be dropped. If, for any reason, you do not turn in one homework assignment, your grade will be determined by the remaining assignments.

Homework is due at classtime on the assigned dates. Late assignments lose 25% of their original value per (MWF) class period. If you are confident that you have a pressing argument for an emergency exemption for these rules, you should resolve the matter with the instructor before the due time.

Appeals of homework grades should be discussed first with the GTF. The most effective procedure is to submit you comments in writing and allow time for a careful reconsideration. In contesting a grade, be aware that it is unlikely that you will make any headway based upon the number of points lost for a particular type of error. The grading will have followed uniform criteria for the entire class; you may not agree with the emphasis but all submissions will have been treated equitably. Thus, expect that changes will be made only if there was an actual mistake in grading. Examples: scores were incorrectly added; or a solution was obviously misread; or you had a different, but justifiable, interpretation of a problem that was missed by the grader.

*Homework submissions must consist entirely of your own work.* Substantive discussion of the homework with other students is inappropriate. If any sources are consulted, they must be fully cited.

**Exams**

Problems on exams are likely to resemble homework problems. In particular, you need not be prepared to parrot definitions or proofs. Of course, familiarity with specific algorithms, sometimes by name, and the techniques underlying their design, verification, and analysis, are intrinsic to the course.

The midterms and final will be closed-book exams. However, two 8½ by 11 pages (or four sides) of notes will be permitted. These are to be stapled to and turned in with the exam. It is permissible to exchange ideas prior to the exam on the content of these pages, but *you must prepare your own set of notes* and these must be written or typed by you (not photo-copied or electronically transferred from the notes of another student and not photocopied from the text or other source). Write, or use a font, as small as you like.

**Extra-credit**

Challenges will be posed, from time to tome, either in class or with the homework assignment. Do not, however, turn them in with regular homework. Give them separately to the instructor or teaching assistant. Typically, extra-credit problems should be turned in within two weeks of their proposal. It is possible you may find that ideas for solutions, or even full solutions, in other texts or reference materials. Again, you are obliged to cite any

Extra credit may also be assigned for superior contribution to class discussions.

**Lectures**

Lectures are intended to supplement rather than duplicate the text. Material may be discussed in class that is not in the text. This is particularly true for the algorithm animation component of the course. In addition, exam and homework problems may be based upon material presented in lectures. It is strongly advised that you regularly attend, take notes, and review them.

**Tentative syllabus for first exam**

Chapters 10, 16, 17.

**Useful review material**

Chapters 1–5.

*Future announcements will appear in the newsgroup* `uo.cs.cis.315`

CIS 315 Syllabus for Study II[84]

| | |
|---|---|
| **Class meetings** | Mon, Wed, Fri, 9:00-9:50 161 Bar |
| **Instructor** | John Lane, 100 Deschutes<br>Office hours: to be announced |
| **Teaching Assistant** | Tom Maly, 200 Deschutes<br>Office hours: to be announced |
| **Teaching Assistant for algorithm animation** | Chris Hundhausen, 231 Deschutes |
| **Prerequisites** | CIS 313, Math 233 |
| **Text** | Cormen, Leiserson, Rivest<br>*Introduction to Algorithms* |
| **Newsgroup** | `uo.classes.cis.315`<br>for assignments, announcements, hints, etc. |
| **Assignments** | Homework problems assigned regularly.<br>One algorithm animation project<br>Optional programming assignment<br>As alternative to final exam |

**Grading**

| | |
|---|---|
| Homework | 21% |
| Algorithm animation | 16% |
| First midterm | 21% |
| Second midterm | 21% |
| Final | 20% |
| Extra credit problems | variable credit |

**Important dates**

| | |
|---|---|
| First midterm | Wed, Apr 23 |
| Second midterm | Mon, May 19 |
| Animation due | Mon, Jun 2 |
| Final exam | Mon, Jun 9 |

*—The rest of the syllabus was identical to that used in Study I—*

---

[84]Certain items in this syllabus have been changed or omitted in order to preserve anonymity.

# Study I Animation Assignments

**All of the assignments during Study I were posted to the newsgroup.**

## Assignment 1

```
Subject: Animation Assignment #1:  Flying Logos


Due:  Wednesday, January 22

The purpose of this assignment is to familiarize you with the Samba
language.  Create an animation in which your name (a text string), along
with other geometric shapes, flies around in one or more animation
windows.  Don't be afraid to experiment with lots of different Samba
commands, in order to gain familiarity with the language.  Try out various
X Windows colors and fonts as well.

In order to become comfortable with the idea of annotating a program, you
are encouraged to create your flying logo by annotating a simple program
with Samba commands.  Your program might use iteration to create
objects and move them around.

The assignment is worth 1% of your grade.  Particularly creative logos may
earn extra credit.  In addition, particularly creative logos may earn
a place in a virtual gallery of animations we'll be setting up.

Please copy your executable Samba file (with your name as a comment on the
first line of the file) by the due date to the

/cs/classes/cis315/assignments/flogo

directory, which is world-writable, but which may only be read by your
instructor.

:-)Chris
```

## Assignment 2

```
Subject: Algorithm Animation Assignment #2, due Wed., Feb. 12

       ANIMATION OF A DIVIDE-AND-CONQUER OR DYNAMIC PROGRAMMING ALGORITHM
                   DUE WED, FEBRUARY 12, 1996 BY 11:59 P.M.
                           WORTH 5% OF YOUR GRADE

For this animation assignment, pick a favorite divide-and-conquer (DAC) or
dynamic programming (DP) algorithm.  You are free to choose any algorithm
that (a) we've talked about in class, (b)  appears in the Cormen text, (c)
is published somewhere else, or (d) you've written yourself.  The only
restrictions are (a) that it _not_ be a sorting algorithm, and (b) that it
employ either the DAC or DP strategy.

Candidate algorithms for the assignment include (but by no means are
restricted to)

(a) quickselect;
(b) the Skyline problem (please e-mail me if you would like a xerox of the
```

three-page description of the algorithm, which appears in
"Introduction to Algorithms: A Creative Approach," by Udi Manber);
(c) the DAC convex hull algorithm; and
(d) the knapsack problem.

Once you've chosen an algorithm, create an animation for the algorithm in Samba. Acting as the narrator, you will present the animation to your classmates on a later date. Thus, your goal, in creating the animation, should be to communicate the fundamental aspects of the algorithm to your classmates, keeping in mind that you will be able to provide the play-by-play for the animation when you ultimately present it. Be creative! Use multiple views, if you feel that they would add to your presentation.

You are encouraged to find partners and work in pairs; keep in mind that more will be expected from pairs than from individuals. For the final assignment, you will be allowed to work in larger groups.

By the deadline, please do the following:

(1) Create your own subdirectory in the /cs/classes/cis315/assignments/a2 directory. Please give that subdirectory the same name as your login name. If you're working in a pair, then name the directory a string containing your login name, an underscore, and your partner's login name (e.g., foo_bar).

(2) In your subdirectory, place (a) your algorithm's source code, (b) an executable of your algorithm, (c) a sample samba trace of your algorithm running on a sample data set, and (d) a README file indicating how to run the algorithm to produce a samba trace. Set your algorithm up so that it will take an input data file as a parameter on the command line. You can define whatever input data file format you want, but please document that format in your README file. Keep in mind that your instructor and GTF may experiment with running your algorithm on alternative data sets. Hence, you should design the algorithm animation so that it will run on an arbitrary input data set.

Your animation's grade will be based on the following criteria (listed roughly in order of importance):

(a) creativity,
(b) effectiveness in communicating the algorithm,
(c) correctness (not in the formal sense, but your instructor and GTF will be checking to make sure that you haven't rigged it so that it only works on a certain data set)

If you have any questions regarding the feasibility of a particular algorithm, or if you are having trouble coming up with a way to illustrate your chosen algorithm, please contact Professor Lane, Tom or me for a consultation.

## Assignment 3

Date: Sat, 1 Mar 1997 09:33:29 -0800
From: Christopher David Hundhausen <chundhau@cs.uoregon.edu>
Newsgroups: uo.classes.cis.315
Subject: Animation Assignment #3 -- Clarification

Many of you have already started this assignment; here's the official posting:

FINAL ALGORITHM ANIMATION ASSIGNMENT
DUE (check the syllabus)
WORTH 15% OF YOUR GRADE

Pick one or more non-sorting algorithms that employ one of the general problem-solving strategies covered in the course (dynamic programming, divide-and-conquer, greedy, etc.). Create a Samba animation of the algorithm(s). Given that this assignment is worth three times as much as the second animation assignment, your animations should go beyond the animations you programmed for the last assignment. For example, you might

* illustrate a proof of an algorithm's correctness (e.g., Kruskal, Prim);

* create side-by-side races of two algorithms that solve the same problem, emphasizing the ways in which the algorithms are similar and different;

* provide multiple perspectives on an algorithm by showing, for example, the call stack, tree of recursive calls, or other important information; or

* animate a _real world application_ of an algorithm, as opposed to the algorithm itself.

Acting as narrators, your group will present the animation to your classmates on a later date. (Note: Due to schedule conflicts with rooms containing projection devices, you will not be asked to present your second animation assignment; however, all or most of you will be asked to present your third animation assignment.) Thus, your goal, in creating the animation, should be to communicate the fundamental aspects of the algorithm to your classmates, keeping in mind that you will be able to provide the should be to communicate the fundamental aspects of the algorithm to your classmates, keeping in mind that you will be able to provide the play-by-play for the animation when you ultimately present it.

You are encouraged to work in groups for this assignment; there is no limit to group size. The larger the group, the more that will be expected of the group.

By the deadline, please do the following:

(1) Create your own subdirectory in the /cs/classes/cis315/assignments/a3 directory. Please give that subdirectory the same name as your login name. If you're working in a group, then name the directory a string containing your login name, an underscore, and your first partner's login name, an underscore, your second partner's name, . . ., an underscore, your nth partner's name. (e.g., foo_bar_bye).

(2) In your subdirectory, place (a) your algorithm's source code, (b) an executable of your algorithm, (c) a sample samba trace of your algorithm running on a sample data set, and (d) a README file indicating how to run the algorithm to produce a samba trace. Set your algorithm up so that it will take an input data file as a parameter on the command line. You can define whatever input data file format you want, but please document that format in your README file. Keep in mind that your instructor and GTF may experiment with running your algorithm on alternative data sets. Hence, you should design the algorithm animation so that it will run on an arbitrary input data set.

# Calls for Participation

From: Christopher David Hundhausen <chundhau@cs.uoregon.edu>
Newsgroups: uo.classes.cis.315
Subject: Invitation to join the algorithm animation mailing list

For those of you who weren't at the Thursday morning session this week, I'd like to invite you to join the algorithm animation mailing list. I'm interested in your experiences with algorithm animation. I'll occasionally be posting brief surveys or requests for volunteers to the list. If you wouldn't mind helping me out, please send me you're e-mail address and I'll add you to the list.

Thanks,

:-)Chris

---

From: Christopher David Hundhausen <chundhau@cs.uoregon.edu>
To: "Algorithm Animation List
Subject: I'd like to hang out with your group!

Thanks again for your interest in my algorithm animation mailing list. As I mentioned earlier, I may send requests and surveys to the list from time to time.

As you know, the second algorithm animation assignment is due next Wednesday (Feb. 12). I'm interested in how you're going about the assignment. Would any of you be willing to let me work with your group? If you think you might be interested, please send me e-mail or talk to me after class ASAP.

Thanks in advance,

:-)Chris

Informed Consent Agreements

## INFORMED CONSENT AGREEMENT: ALGORITHM VISUALIZATION STUDY

You are invited to participate in a research study being conducted by Christopher Hundhausen, a Ph.D. student in the University of Oregon's Department of Computer and Information Science. For my dissertation research, I am studying the use of algorithm visualization in undergraduate algorithms courses. The goal of the research is to develop a theory of effectiveness that addresses the use of algorithm visualization artifacts within this context. Because you are presently enrolled in Computer and Information Science 315 (*Introduction to Algorithms*), you were selected as a possible participant in this study.

You may elect to participate in the study in at least five ways:

1. You may volunteer to allow me to videotape your storyboard presentation and/or animation presentation. If you elect to allow me to videotape your animation sessions, you will be asked to sign a separate form (*Consent Agreement for Algorithm Animation Presentations*).

2. You may volunteer to allow me to observe and/or participate in either your development of a storyboard presentation, your development of a Samba animation, or both. As I observe and participate, I may take notes, and I may ask permission to audiotape interviews or group conversations. If you elect to allow me to audiotape interviews or conversations during my participant observation sessions, you will be asked to sign a separate form (*Consent Agreement for Audiotaping*).

3. You may volunteer to allow me to audiotape an interview with you about your experiences in developing either your storyboard presentation, or your Samba animation. Such interviews will usually take from 5 – 45 minutes. If you elect to allow me to audiotape an interview with you, you will be asked to sign a separate form (*Consent Agreement for Audiotaping*)

4. You may volunteer to allow me to photograph (or keep) your storyboard, or to keep a copy of your Samba animation (including all source code files used to generate it). If you consent to either of these, you will be asked to sign a separate form (*Consent Agreement for Release of Storyboard or Samba Animation*).

5. You may volunteer to allow me to maintain copies of the diary you kept, and the test you developed, for the course Algorithm Animation Assignment. If you elect to allow me to maintain copies of these, you will be asked to sign a separate form (*Consent Agreement for Release of Animation Assignment Diaries and Tests*).

Because (1), (4), and (5) do not require you to engage in activities beyond those required for course assignments, they do not involve any risks beyond the minimal risks associated with participating in any university course. The risks associated with (2) and (3) are deemed to be minimal; you will not be subjected to any pain or stress beyond that normally encountered in everyday life. This research aims to improve the effectiveness of course activities involving algorithm visualization. However, I cannot guarantee that you personally will receive any benefits from this research.

Any information that (a) is obtained in connection with this study, and that (b) can be identified with you, will remain confidential and will be disclosed only with your permission or as required by law. Within all published written reports and oral presentations of this study's results, participant identities will be kept confidential by assigning each participant

a code. All data will be marked and identified using this code. During the present academic term, a separate list that associates names with codes will be maintained for the purposes of contacting participants for follow-up interviews; however, this list will be destroyed within six months of the end of the academic term. All audiotapes, videotapes, and copies of your assignments will be kept in a secure location at all times during and after the study. Only members of the Interactive Systems Group and other researchers associated with this study shall be allowed access to them.

Your participation is voluntary. Your decision whether or not to participate will not affect your grade in the CIS 315 course in which you are presently enrolled, nor will it involve a penalty or loss of benefits to which you are otherwise entitled. If you decide to participate, you are free to withdraw your consent and discontinue participation at any time without penalty.

If you have any questions, please feel free to contact me, Christopher Hundhausen (346–4425), or my dissertation advisor, Professor Sarah Douglas (346–3974) in the University of Oregon's Computer and Information Science Department. If you have questions regarding your rights as a research participant, contact Human Subjects Compliance, University of Oregon, Eugene, OR 97403, (541) 346-2510. You will be given a copy of this form to keep.

Your signature indicates that you have read and understand the information provided above, that you willingly agree to participate, that you may withdraw your consent at any time and discontinue participation without penalty, that you will receive a copy of this form, and that you are not waiving any legal claims, rights or remedies.

______________________________ ____________________

Signature Date

## CONSENT AGREEMENT FOR VIDEOTAPING OF ALGORITHM ANIMATION PRESENTATIONS

I have received an adequate description of the purpose of, and procedures for, videotaping the storyboard presentations and Samba animation presentations sessions during the course of the proposed research study. I give my consent to be videotaped during (initial all that apply)

_______my storyboard presentation (Date:_______________________)

_______my Samba animation presentation (Date:_______________________)

_______at other times during the session in which I present my storyboard

(Date:_____________________)

_______at other times during the session in which I present my Samba animation

(Date:_______________)

Further, I agree to allow the videotape(s) to be viewed by members of the Interactive Systems Group in the University of Oregon's Department of Computer and Information Science. (If you agree to allow Christopher Hundhausen to show excerpts of your presentation(s) at professional conferences and academic colloquia, for the express purpose of reporting the results of the research project described in the attached *Informed Consent Agreement*, please initial here:_______.) I understand that all information will be kept confidential and will be reported in an anonymous fashion, and that the videotapes will be kept on file for several years, in order that they may be available for further analysis in the future. I further understand that I may withdraw my consent at any time.

_______________________________________  _________________________

Signature  Date

## INFORMED CONSENT AGREEMENT FOR AUDIOTAPING

I have received an adequate description of the purpose of, and procedures for, audiotaping interviews and conversations during the course of the proposed research study. First, I give my consent to be audiotaped. Second, I agree to allow the audiotape to be reviewed by members of the Interactive Systems Group in the University of Oregon's Department of Computer and Information Science. Third, I agree to allow the audiotapes to be transcribed for possible use in any published reports on the research described in the attached *Informed Consent Agreement*. Fourth, I understand that all information will be kept confidential and will be reported in an anonymous fashion, and that the audiotapes will be erased after an appropriate period of time after the completion of the study. Finally, I understand that I may withdraw my consent at any time.

______________________________ ____________________

Signature Date

## CONSENT AGREEMENT FOR RELEASE OF ALGORITHM ANIMATION ASSIGNMENT

I have received an adequate description of the purpose of allowing Christopher Hundhausen to maintain copies of the various components of my algorithm animation assignment (storyboard, Samba animation, assignment diary, assignment test). I give my consent (please initial all that apply)

______1. to allow Christopher Hundhausen to photograph and/or digitize the storyboard I developed for the algorithm animation assignment in CIS 315 (Spring term, 1997) and to allow Christopher Hundhausen and members of the Interactive Systems Group in the University of Oregon's Department of Computer and Information Science to review the reproductions.

______2. to allow Christopher Hundhausen to maintain a copy of the Samba animation (including all source code used to generate it) I developed for the algorithm animation assignment in CIS 315 (Spring term, 1997), and to allow Christopher Hundhausen and members of the Interactive Systems Group in the University of Oregon's Department of Computer and Information Science to review your animation.

______3. to allow Christopher Hundhausen to maintain a copy of the assignment diary that I developed for the algorithm animation assignment in CIS 315 (Spring term, 1997), and to allow Christopher Hundhausen and members of the Interactive Systems Group in the University of Oregon's Department of Computer and Information Science to review the diary.

______4. to allow Christopher Hundhausen to maintain a copy of the assignment test that I developed for the algorithm animation assignment in CIS 315 (Spring term, 1997), and to allow Christopher Hundhausen and members of the Interactive Systems Group in the University of Oregon's Department of Computer and Information Science to review the test.

______5. to allow Christopher Hundhausen to associate my name with (a) digitized representations of my storyboard and (b) my Samba animation in any published reports, professional conference presentations, or academic colloquia on the study described in the attached *Informed Consent Agreement*.

First, I understand that those components of my algorithm animation assignment that I release to Christopher Hundhausen will be kept in a secure place, so that only those parties whom I have granted access to them may access them. Second, I agree to allow digital representations of my storyboard, still frames of my Samba animation, and/or excerpts of my algorithm animation assignment diary and algorithm animation assignment test to be reproduced in published reports of the study described in the attached *Informed Consent Agreement*. Third, if I have initialed (2), I also agree to allow my Samba animation, or excerpts thereof, to be presented at professional conferences or academic colloquia for the purpose of reporting the results of the study described in the attached *Informed Consent Agreement*. Fourth, I understand that all information will be kept confidential, and that, unless I have initialed (3), all information will be reported in an anonymous fashion. Fifth, I understand that photographs of my storyboard, digital representations of my storyboard, my Samba animation, my algorithm animation assignment diary, and/or my algorithm animation assignment test will be kept on file for several years, in order that they may be available for further analysis in the future. Finally, I understand that I may withdraw my consent at any time.

______________________________      ____________________

Signature      Date

## Study I E-Mail Surveys

From: Christopher David Hundhausen <chundhau@cs.uoregon.edu>
To: "Algorithm Animation List"

I would appreciate your assistance with my dissertation research on the use of algorithm visualization in undergraduate algorithms courses. This research will help me to assess the value of algorithm visualization, and to develop more effective uses of it, in undergraduate algorithms courses.

To help me, all you need to do is complete this short survey, which should take approximately 5 - 10 minutes. If you do not wish to participate, simply delete the survey. Responses will be completely anonymous; your name will not appear anywhere on the survey. Completing and returning the questionnaire constitutes your consent to participate.

Keep this message for your records. If you have any questions regarding the research, contact me, Christopher Hundhausen (chundhau@cs.uoregon.edu) or my dissertation advisor, Sarah Douglas (douglas@cs.uoregon.edu). If you have any questions regarding your rights as a research participant, please contact Human Subjects Compliance at the University of Oregon, (541) 346-2510. Thank you again for your help.

*************************************************************************

ALGORITHM ANIMATION SURVEY #1

1) What was your overall impression of this assignment? What did you like about it, and what didn't you like about it?

2) If you had to break up the activities you performed for this assignment into distinct phases, what would those phases be, and what did you do during each of the phases?

3) About how much time did you spend on the assignment? Please break your time estimate down further by indicating the amount of time you spent on each of the phases you listed in (2).

4) What do you feel you learned from developing an algorithm animation, if anything?

5) Do you have any suggestions for future algorithm animation assignments?

THANK YOU FOR YOUR TIME! If you have any other comments, feel free to attach them to the end of this survey.

To: "Algorithm Animation List"
Subject: A quick survey on the final animation assignment

Hi all,

If you have some time, I'd appreciate it if you could take a few minutes to respond to this brief survey regarding the final animation assignment. This will most likely be the last posting to this group. Thanks so much for volunteering to help me out; your responses to the first survey were really helpful.

Once again, I will not associate any of your responses with your name; that is, you can answer the survey anonymously.

:-)Chris

==========================================================================

ALGORITHM ANIMATION SURVEY #2

1) Now that you have gained a fair bit of experience with algorithm animation, what do you like about it? What don't you like? Do you feel that you've benefited from the assignments, or have they been a nuisance?

2) How does your experience with the final animation assignment compare with your experience with the second animation assignment? Did you approach the problem in a similar way? Did you do anything different? Why?

3) What aspects of the animation assignments do you feel are most beneficial? What aspects of the assigment could you do without?

4) Do you have any comments on Samba that I could pass along to its developer? Are there any features that you wish you'd had? Was Samba easy to use? What could be improved upon? Now's your chance to speak your mind!

***THANKS FOR YOUR PARTICIPATION!***

Study II Animation Assignment

## CIS 315 Algorithm Animation Assignment Spring, 1997

**Note:** For this assignment, you are encouraged to work in pairs, but groups of one and three are also permissible. The larger the group, the more that will be expected of the group.

## PHASE 1: ANIMATION PROSPECTUS

**Choose an algorithm.** Begin by picking an algorithm or group of algorithms that you would like to present to your classmates in the form of a graphical animation. With the exception of sorting algorithms, you may choose any algorithm(s) covered in class, or any algorithms related to those covered in class. Candidate algorithms include any divide-and-conquer, dynamic programming, greedy, and graph searching algorithms.

**Choose a focus.** Next, choose a focus for your presentation. To earn a C, your presentation should minimally illustrate your algorithm's behavior—that is, how it works. To do so, you will need to choose carefully not only what aspects of the algorithm are important, but also what input data will illuminate those aspects.

To earn more than a C, your presentation should take on one or more issues that go beyond your algorithm's procedural behavior, such as

- correctness (Why does it work?),
- efficiency (How quickly does it work?), and
- applicability (How can it be applied to a real-world problem?).

For example, you might

- compare a naïve algorithm to a clever algorithm (e.g., the naïve recursive approach to computing the $n^{th}$ Fibonacci number versus the dynamic programming approach), in order to illustrate the superiority of the clever algorithm;
- present an animated "proof" of an algorithm's correctness;
- present an algorithm's efficiency graphically;
- compare two algorithms that solve the same problem (e.g., Kruskal's vs. Prim's minimum spanning tree algorithms); or
- illustrate the application of an algorithm to a real-world problem (e.g., show how one can apply the topological sort to a real-world scheduling problem).

**Develop a storyboard presentation.** Now, develop an interactive storyboard presentation with your chosen focus (see Guidelines for Developing a Storyboard Presentation; a PostScript version is available in `/cs/classes/cis315/assignments/storyboard_guidelines.ps`). You will present your storyboard to your instructor, and to a small group of your classmates, during one of the prospectus presentation sessions to be scheduled during the two week period from May 5th to 16th, with a possible weekend session on May 10th or 11th. The target length of your presentation is **ten minutes**. Note that, while it will certainly be possible to

produce your storyboard "on the fly" during the course of your presentation (on a whiteboard, for example), it is highly recommended that you come to your presentation with at least a partially-constructed storyboard, which will help you to structure your presentation, and which you will be able to mark up based on the feedback you receive from your instructor and audience.

**Keep a diary.** During the course of developing your storyboard presentation, each member of your group should keep a separate **diary** of her or his experiences. To structure your entries, you are encouraged to use the **diary form** available (as PostScript) in `/cs/classes/cis315/assignments/diary_form.ps`. (Print out as many copies as needed). Since the purpose of the diary is not only to help us improve future animation assignments, but also to help you keep track of and reflect on your own development efforts, it is highly recommended that you attempt to log your activities as you go along—not as an afterthought before you hand in the assignment! If you are working in a group of two or three, also explain in your diary (on an attached sheet) how your group members' efforts were divided (Who did what?), and provide a brief critique of your group members (Did they carry their weight?). Note that, while your diary cannot hurt your grade, a thoughtful diary may help it.

**Develop a mock test.** Along with your diary, please develop **a brief test** (around 5 questions) that you would expect people who view your presentation to do well on. One test per group is sufficient.

## PHASE 2: SAMBA ANIMATION

Now implement your storyboard as a Samba animation, taking into consideration the suggestions and feedback you received on the storyboard presentation you developed during Phase 1. Acting as narrators, you and your group members will present the animation to the class during the week of **June 2 – 6**. For your presentation, be prepared to explain your animation as you step through it, and to answer questions from the audience. A device that projects a Sun Sparcstation screen onto a large screen will be provided for your presentation.

As before, each member of your group should keep a separate **diary** according to the guidelines stated above. Also, please revisit the **test** that you developed for Phase 1 of the assignment, and make any modifications necessary to bring it up to date with your Samba animation.

In addition, **by midnight on June 2**, you should do the following:

- Create your own subdirectory in the `/cs/classes/cis315/assignments/a1` directory. Please give that subdirectory the same name as your login name. If you're working in a group, then separate your group members' login names with underscores (e.g., `you_me_her`).
- In your subdirectory, place (1) one or more Samba trace files (each of which should have a `.samba` file extension) containing sample animations, (2) all C++ (or whatever programming language you used) source code used to generate the Samba traces, and (3) a `README` file indicating (a) what algorithm you're illustrating, (b) what each sample trace file illustrates; (c) how to arrange the windows of your animation for optimal viewing, and (d) the optimal speed at which to view the sample files.

Study II: Guidelines for Developing a Storyboard Presentation

**CIS 315** **Spring, 1997**

# Guidelines for Developing a Storyboard Presentation

What is a storyboard? Introduced by the cartoon animation industry in the 1930s, a storyboard presents just the *key frames* of the animation—that is, the snapshots that capture the main events. With your help, the audience can then fill in the rest of the details. For example, if you were to storyboard a scene in which the Roadrunner slyly leads the Coyote off a cliff, you might include four snapshots:

- one in which the Coyote is chasing the Roadrunner atop a mountain;
- one in which the Roadrunner ducks behind a rock wall just before a steep cliff;
- one in which the Coyote has just run off the cliff and is temporarily suspended in mid-air; and
- one in which the Coyote lands flat on his face in the canyon below.

Using a storyboard containing these four key snapshots, you could explain to your colleagues how the scene would unfold, and your colleagues could provide feedback and suggestions. Likewise, your algorithm animation storyboard should assist you in communicating to your instructor and audience the manner in which your algorithm animation will unfold. Your storyboard presentation is your chance to discuss your ideas, to ask questions or advice, and to receive useful feedback that you can incorporate into your final animation.

**Suggestions for constructing storyboard presentations.** To construct your storyboard, you are free to use any resources you deem appropriate, including (but not limited to)

- paperboard
- construction paper cut-outs
- colored markers
- colored pencils
- transparencies (an overhead projector will be available)
- watercolor paint
- whiteboard markers (a whiteboard will be available)

Be creative! For example, you might come to your presentation with a large piece of paperboard partitioned into several sections (for example, one reserved for each iteration of your algorithm's loop), and with paper cut-outs of the key objects in your animation. For each phase of your animation, you could place a set of cut-outs on the storyboard, and demonstrate the animation by sliding the cut-outs across the paperboard. Alternatively, you might choose to build a storyboard out of overhead transparencies, on which you could actually draw (or which you could slide cut-outs across) to demonstrate movement. In any case, make sure that your storyboard is *sufficiently large* for a medium-size group (5 – 15 people) to view comfortably.

# Study II Diary Form

**CIS 315** **Algorithm Animation Diary Form** **Spring, 1997**

**Name:**______________________ **Page** _________ **of** ___________

| DATE: | START TIME: | FINISH TIME: | ELAPSED TIME: |
|---|---|---|---|
| DESCRIBE ACTIVITIES: | | | |
| PROBLEMS ENCOUNTERED: | | | |
| COMMENTS: | | | |

| DATE: | START TIME: | FINISH TIME: | ELAPSED TIME: |
|---|---|---|---|
| DESCRIBE ACTIVITIES: | | | |
| PROBLEMS ENCOUNTERED: | | | |
| COMMENTS: | | | |

| DATE: | START TIME: | FINISH TIME: | ELAPSED TIME: |
|---|---|---|---|
| DESCRIBE ACTIVITIES: | | | |
| PROBLEMS ENCOUNTERED: | | | |
| COMMENTS: | | | |

*(If you need more space, feel free to use the reverse side.)*

# APPENDIX D

# EXPERIMENT MATERIALS

Preliminary Interest Questionnaire

## Algorithm Animation Study: Preliminary Interest Questionnaire

*Chris Hundhausen, a Ph.D. student in the CIS department, will be conducting a research study in conjunction with the winter term, 1998 offering of CIS 315 (Algorithms). If you decide to participate in the study, you would use algorithm animation to explore an algorithm, and do a series of exercises designed to assess your understanding of the algorithm. You would spend around 5 hours on the study, and be paid $50. In addition, since you would be learning about an algorithm covered in CIS 315, your participation may help you with CIS 315.*

I intend to enroll in CIS 315 for winter term, 1998 (circle one):

DEFINITELY MAYBE NO

If you chose Definitely or Maybe, please answer the following two questions:

Please rate your level of interest in participating in this study (circle one):

| NO INTEREST | LITTLE INTEREST | NEUTRAL | INTERESTED | VERY INTERESTED |
|---|---|---|---|---|

Please estimate the likelihood that you will participate in this study (circle one):

| NO CHANCE | REMOTE POSSIBILITY | 50-50 CHANCE | GOOD CHANCE | DEFINITELY WILL |
|---|---|---|---|---|

If you wish to be contacted further regarding this study, please provide the following contact information:

Name:________________________________E-mail:____________________

Background Information Questionnaire

## ALGORITHM ANIMATION STUDY: BACKGROUND INFORMATION

*For Experimenter Use Only*

*Participant Code:*____________

Name:______________________________ E-mail:____________________________

Major: _______________________ Class standing: Fr So Jr Sr Other:________

CIS Courses taken (list 300-level and above only):______________________________

____________________________________________________________________

Exact Overall GPA (to two decimal places, as of 1/99):_________________

Have you ever used algorithm visualization before? YES NO

If you answered "YES" above, briefly describe your use(s) of it:

Informed Consent Agreement

# INFORMED CONSENT AGREEMENT: ALGORITHM ANIMATION STUDY

You are invited to participate in a research study being conducted by Christopher Hundhausen, a Ph.D. student in the University of Oregon's Department of Computer and Information Science. For my dissertation research, I am studying the use of algorithm animation in undergraduate algorithms courses. The goal of the research is to develop a theory that accounts for the learning benefits of algorithm animation within this context. Because you filled out a preliminary interest questionnaire that I handed out in a CIS 313 or CIS 315 lecture, you were selected as a possible participant in this study.

Your participation in this study will involve using visualization technology to explore a computer algorithm, and completing a series of exercises designed to evaluate your understanding of the algorithm. The study will last around five hours. So that I can better understand your activities, I may observe you and take notes as you engage in some of these activities.

Because none of these activities differs from those in which you would engage in a typical course on algorithms, they do not incur any risks beyond the minimal risks associated with participating in a university course. You will receive $50 for completing all of the activities in this study. In addition, your participation in these activities could help you with the CIS 315 course in which you are presently enrolled.

Any information that (a) is obtained in connection with this study, and that (b) can be identified with you, will remain confidential and will be disclosed only with your permission or as required by law. Confidentiality will be safeguarded by assigning each participant a code. All data will be marked and identified using this code. During the present academic term, a separate list that associates names with codes will be maintained for the purposes of contacting participants; however, this list will be destroyed within on year of the end of the academic term. All paperwork and computer files associated with this study will be kept in a secure location at all times during and after the study. Only members of the Interactive Systems Group and other researchers associated with this study shall be allowed access to them.

Written manuscripts and oral presentations of this study's results will most often report only aggregate data. In cases in which individual data are reported, participant identities will be kept confidential through the use of pseudonyms.

Your participation is voluntary. Your decision whether or not to participate will not affect your grade in the CIS 315 course in which you are presently enrolled, or in any other university course. In addition, your decision whether or not to participate will not involve a penalty or loss of benefits to which you are otherwise entitled. If you decide to participate, you are free to withdraw your consent and discontinue participation at any time. If you discontinue participation prior to completing all of the study activities, you will still be compensated at the rate of $10 per hour of participation, not to exceed $50 total. (Time spent

attending preliminary orientation and screening meetings, along with time spent filling out preliminary paperwork, is not subject to compensation.)

If you have any questions, please feel free to contact me, Christopher Hundhausen (346–4425), or my dissertation advisor, Professor Sarah Douglas (346–3974) in the University of Oregon's Computer and Information Science Department. If you have questions regarding your rights as a research participant, contact Human Subjects Compliance, University of Oregon, Eugene, OR 97403, (541) 346-2510. You will be given a copy of this form to keep.

Your signature indicates that you have read and understand the information provided above, that you willingly agree to participate, that you may withdraw your consent at any time and discontinue participation without penalty, that you will receive a copy of this form, and that you are not waiving any legal claims, rights or remedies.

______________________________ ____________________

Signature Date

Instructions: Active Viewing Group

## INSTRUCTIONS: ALGORITHM ANIMATION LEARNING EXERCISE

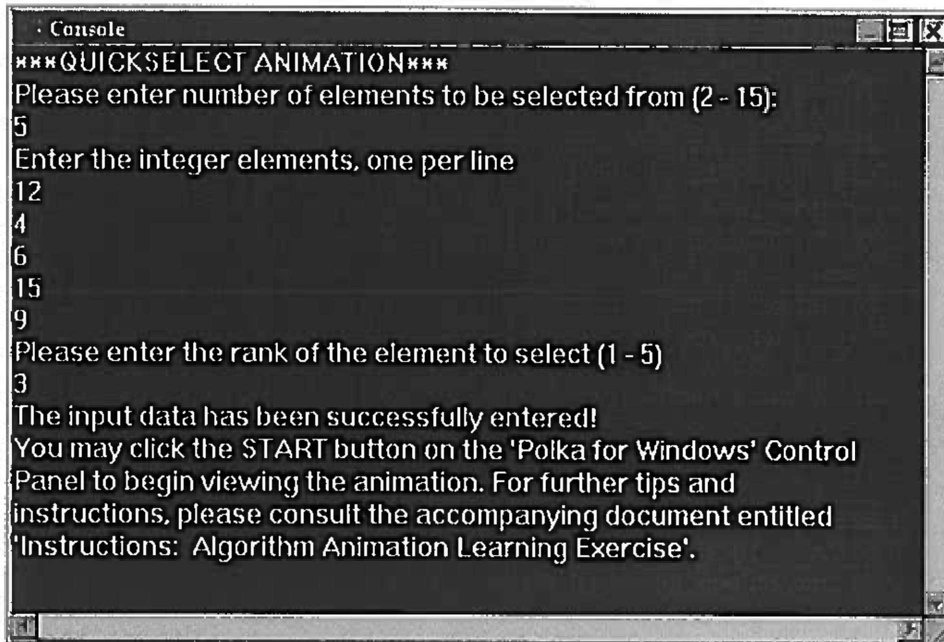
**Learning QUICKSELECT.** In the first part of the study, you'll spend two and a half hours learning about how the QUICKSELECT algorithm works. The QUICKSELECT algorithm, as you may recall from a recent CIS 315 lecture, uses the same divide-and-conquer strategy used by QUICKSORT. However, instead of sorting a list of elements, QUICKSELECT finds the $i^{th}$ smallest element in a list. (Your CIS 315 textbook calls this the $i^{th}$ *order statistic*.) Attached you'll find the pages from your textbook that explain the algorithm, as well as a pseudocode description of the algorithm.

**A note on the PARTITION procedure.** It is important to note that the Partition procedure presented in the attached pseudocode differs from the version of Partition presented in your textbook. The biggest difference is that the Partition that you'll be exploring in this study always selects as the pivot element the *first* element in the array to be partitioned. In contrast, your textbook's version of PARTITION chooses a *random* pivot element. Therefore, even if you feel you know the Partition procedure already, make sure that you study the Partition procedure presented in the attached pseudocode. It is likely to differ, at least slightly, from the versions of Partition that you have seen before.

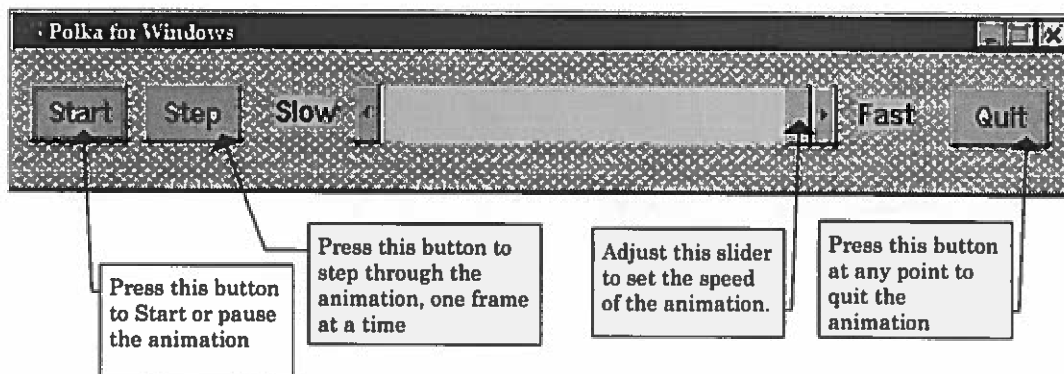
**The QUICKSELECT Animation.** This two-and-a-half hour study session does not just involve reading pseudocode and textbook pages, however. Rather, you are also asked to explore the QUICKSELECT algorithm by viewing and interacting with a computer-based animation.

The animation you'll be exploring explains the QUICKSELECT algorithm through *the Story of the Forest Rangers*. Head Ranger Rhonda has just been notified by her bosses at the National Forest Service that a stand of trees is to be chopped down. However, Rhonda is also told that $i^{th}$ shortest tree in the stand will be spared. With the help of her trusty assistants, Junior Rangers Rodham and Rachel, Rhonda proceeds to apply the QUICKSELECT algorithm to the stand of trees in order to find the tree to be spared. Before the animation begins, you are asked to input the number of elements (trees) to select from, as well as to input the individual values (heights) of the elements (trees). You are then asked to input the rank of the tree to select—that is, the rank of the tree to spare from the clear-cut. You can then run the animation and watch Rhonda, Rodham, and Rachel execute the QUICKSELECT algorithm on your input data set.

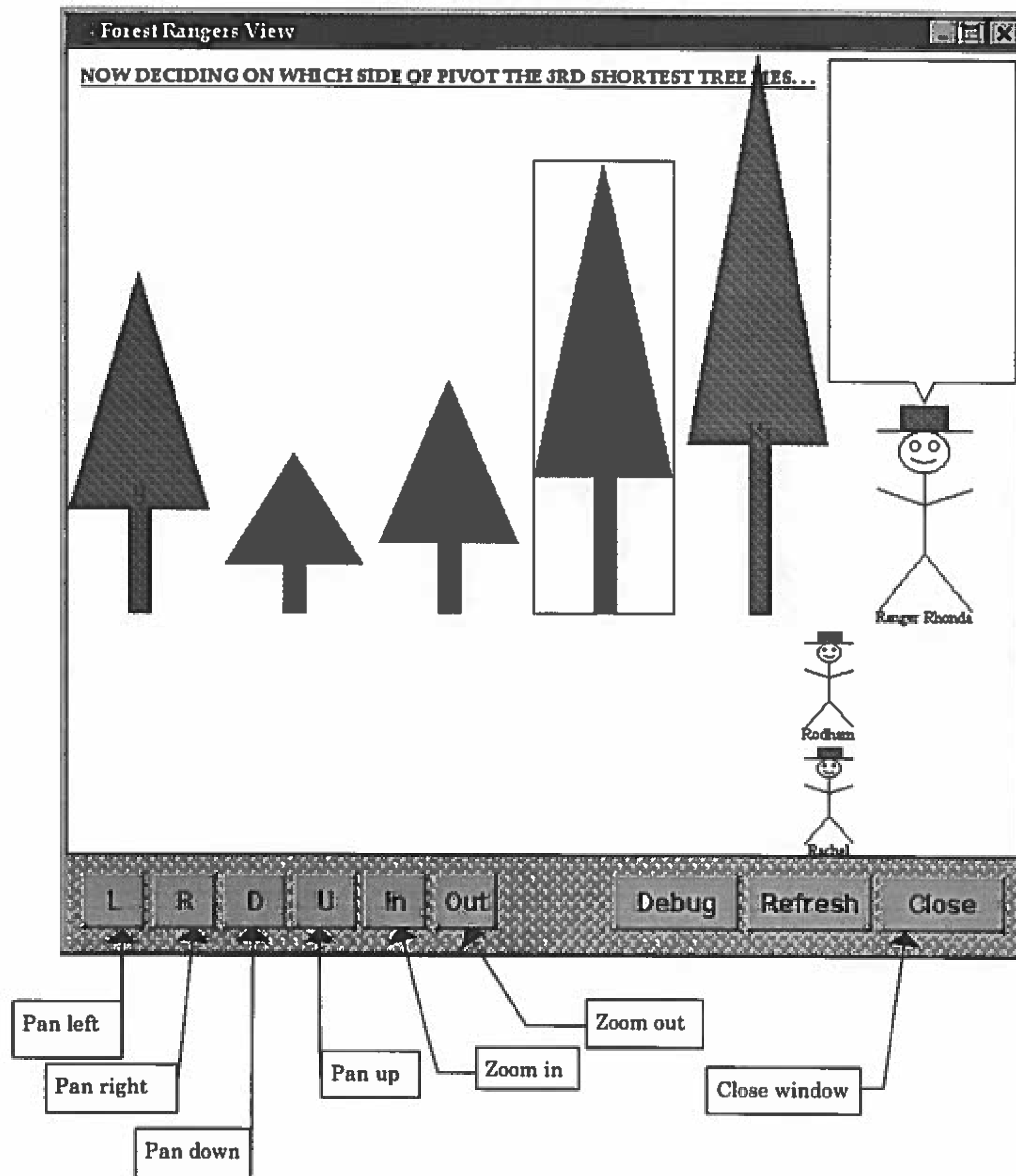
**Elements of the QUICKSELECT Animation.** The QUICKSELECT animation consists of five windows, each of which is illustrated and explained on the following pages.




*The Console window.* Each time you start the animation, the first thing you'll do is enter the input data through the *Console* window. You'll be prompted for the number of elements in the array, the values of the actual array elements, and the rank of the element to select. Once you've entered these values, the console window will report that you are now free to use the *Polka Control Panel* to execute the animation. (Polka, by the way, is the name of the animation package used to implement the animation.)

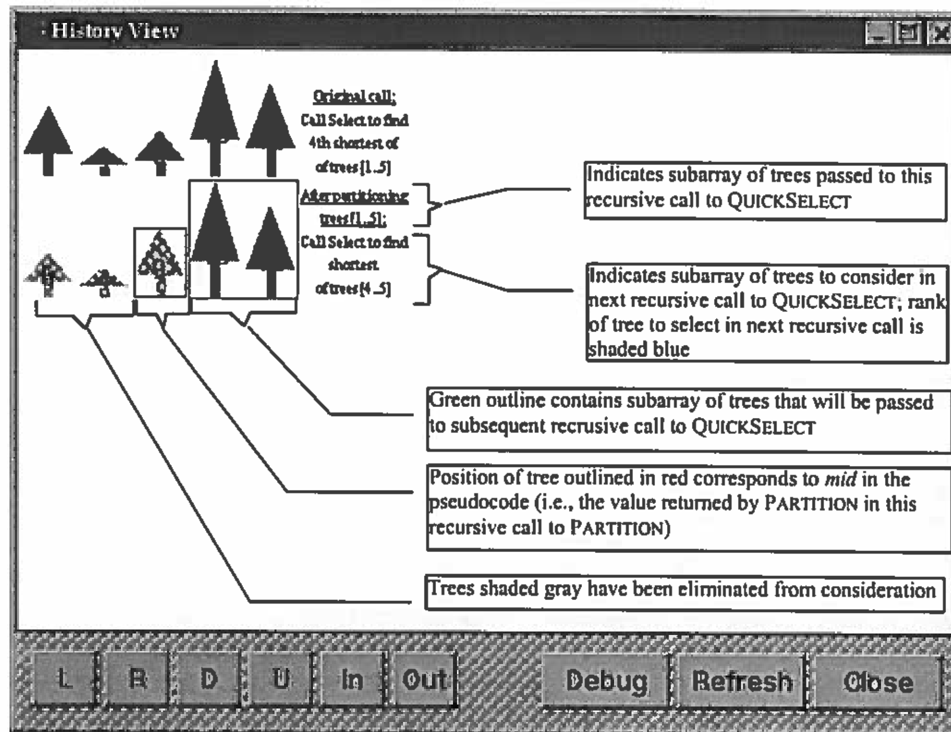



*The Polka Control Panel.* Use the *Polka Control Panel* to control the execution of the animation. The window presents a "tape recorder" style interface, through which you can start, pause, and step through the animation, as well as adjust the animation's execution speed.




*The Forest Rangers window.* In this window, the main action of the animation unfolds. Rhonda introduces the plot of the story, and, with the help of her assistants, Rodham and Rachel, walks you through the execution of the QUICKSELECT algorithm for the input data that you specified. Each ranger has a speaking bubble; you know that a ranger is talking when her or his bubble flashes. As you watch the animation, pay close attention, especially at first, to what the rangers are saying; the animation won't make much sense otherwise. In addition, an explanatory caption across the top of the window summarizes each key step of the algorithm as it unfolds; refer to this caption for help in coordinating the action of the animation with the pseudocode.

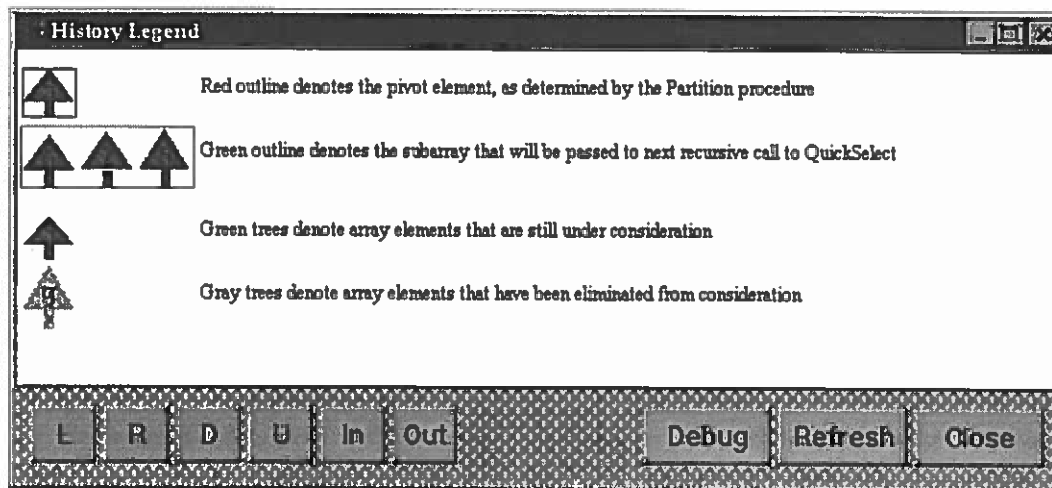
**Note:** All view windows provide buttons at the bottom of the view that you can use to zoom and pan the view. See the labels above for descriptions of each of the important buttons.




*The History window.* Whereas the Forest Rangers window presents a "play-by-play" of the algorithm's key events, the *History* window presents a summary of all of the key algorithm steps that have happened so far. Thus, if it turns out that you miss some action, you can consult the History window for a summary of what you missed.

Each *row* in the *History* window summarizes the key data values of, and decisions made within, one recursive call to the QUICKSELECT algorithm. The arrangement of the trees within the row indicates the state of the array *after* the call to Partition within that recursive call. The annotated window presented above summarizes the notation and shading conventions used by the *History* window.

Note: For larger array sizes, some tree rows will be initially out of view. Use the *D* (pan Down) button to scroll downwards, and the *U* (pan Up) button to scroll upwards.




*The History Legend window.* This window furnishes a legend of the color scheme and notation used in the *History* view. In conjunction with the description of the *History* window discussed above, the *History Legend* window will help you interpret the notation used in the *History* window.

How to use the QUICKSELECT Animation. To launch the QUICKSELECT animation, double-click on the "QuickSelect Animation" icon, which appears on the desktop of your computer:

Now walk through the following four steps:

*Specify your input data.* Think carefully about what kinds of input data will help you to understand the subtleties of how the algorithm works. Use the *Console* window to input a set of data, keeping in mind the following two constraints:

You may only run the animation on arrays of at most 15 elements.

The values of the array elements must fall between 1 and 100.

*Slowly step through the animation.* Now minimize the *Console* window, making sure that the *Forest Rangers* window is fully visible. Set the speed slider on the *Polka Control Panel* to an appropriate speed, and click the *Start* button to begin the animation. Especially the first time you watch the animation, make sure that the speed of the animation is slow enough to enable you to read all of what the rangers are saying; if the action moves by too quickly and you can't read the speaking bubbles, the animation will make little sense. (Note that when the outline of a ranger's speaking bubble flashes, you know that the ranger is speaking.) If the animation is going by too quickly, hit the *Pause* button and readjust the speed. Alternatively, you may hit the *Pause* button and then use the *Step* button to slowly step through the animation, one frame at a time.

*Study the History view.* When the animation is done, click on the *History* window, which is initially obscured by the *Forest Rangers* window. (To view the *History* window, you may want

to minimize the *Forest Rangers* window.) Now take a few minutes to study the "History" view, which summarizes and explains the algorithm's key steps. By studying the series of tree rows, you can obtain a global picture of the way the algorithm operated on the input data.

To interpret the *History* view, see the documentation above, which explains how the *History* view represents the algorithm's key steps. In addition, you may find the *History Legend* window (obscured behind the *History* window) helpful as you study the *History* window.

*Quit the animation.* When the *Forest Rangers* view has run to completion, and when you are done studying the *History* view, you may close the animation by clicking on the *Quit* button in the *Polka Control Panel.* At this point, you can start the animation again as you did before—by double-clicking on the "QUICKSELECT Animation" icon.

Using the QuickSelect Animation as a learning aid. Keep in mind that, in this part of the study, *your main objective is to develop a solid understanding of how the QuickSelect algorithm, including the Partition procedure, works.* Upon completing this part of the study, you should be able to

perform a detailed trace of the QUICKSELECT algorithm (including the PARTITION procedure) operating on any input data set; and

remember the QUICKSELECT algorithm (including the PARTITION procedure) well enough to implement it in a programming language, if you had to.

To refine your understanding of how the QUICKSELECT algorithm works, follow the four steps outlined above to study the algorithm's behavior for at least five input data sets. (Feel free to run the animation on additional input data sets if you have time.) As you become more familiar with the animation, you will probably be able to increase the animation speed, since you may not have to read the speaking bubbles as carefully; however, be sure that you understand what's going on. Also, make sure to choose your input data sets carefully, so that you cover a broad range of possible cases.

Taking a break, and asking questions. Feel free to take a break at any time. (Note that you will be given a half hour lunch break two hours into this session.) If you have questions, raise your hand and I'll come around to help you. When time is called, immediately quit the execution of the animation that you are presently studying, and await further instruction.

Instructions: Self-Construction Group

# INSTRUCTIONS: ALGORITHM ANIMATION CONSTRUCTION EXERCISE

**Learning QuickSelect**. In the first part of the study, you'll spend two and a half hours learning about how the *QuickSelect* algorithm works. The QUICKSELECT algorithm, as you may recall from a recent CIS 315 lecture, uses the same divide-and-conquer strategy used by QUICKSORT. However, instead of sorting a list of elements, QUICKSELECT finds the $i^{th}$ smallest element in a list. (Your CIS 315 textbook calls this the *$i^{th}$ order statistic*.) Attached you'll find the pages from your textbook that explain the algorithm, as well as a pseudocode description of the algorithm.

**A note on the PARTITION procedure**. It is important to note that the PARTITION procedure presented in the attached pseudocode differs from the version of PARTITION presented in your textbook. The biggest difference is that the PARTITION procedure that you'll be exploring in this study always selects as the pivot element the *first* element in the array to be partitioned. In contrast, your textbook's version of PARTITION chooses a *random* pivot element. Therefore, even if you feel you know the PARTITION procedure already, make sure that you study the PARTITION procedure presented in the attached pseudocode. It is likely to differ, at least slightly, from the versions of PARTITION that you have seen before.

**Create a *homemade animation***. This two-and-a-half hour study session does not just involve reading pseudocode and textbook pages, however. Rather, as you study the pseudocode and textbook pages, you should design and ultimately construct your own *homemade animation* of the algorithm. The raw materials for your homemade animation are the art supplies in front of you—construction paper, scissors, glue, and colored pens. The overall goal of your homemade animation is to illustrate, as clearly as possible, how the QUICKSELECT algorithm works. The following questions should help you to get started with this exercise:

1. *What is important about QUICKSELECT?* In designing your homemade animation, first think carefully about what is important to understanding how the QUICKSELECT algorithm works. What are the fundamental operations of the algorithm? How do they alter the algorithm's data? In the case of the QUICKSELECT algorithm, you'll need to consider not only the QUICKSELECT algorithm itself, but also the PARTITION algorithm on which it relies.

2. *What to draw? What to say?* Often, when we explain how an algorithm works, we pick a sample set of input data, and then we *simulate* the algorithm's procedural behavior on that sample input data. We often do so by drawing and modifying sketches , providing a "play-by-play" narrative along the way. For example, if we had to explain a sorting algorithm to someone else, we might sketch out the array after each pass of the algorithm, and we might use narration, gestures, and additional marks on the array to explain how the algorithm gradually places the array elements in order.

3. *What story to tell?* We might also come up with a *story* to explain the algorithm. The story might help us to remember how the algorithm works. Events in the stories might map to significant operations in the algorithm. In addition, the story might create a backdrop that motivates the use of the algorithm. For example, if we wanted to explain a graph searching algorithm, we might construct a story in which a parent is attempting systematically to find a lost child in supermarket aisles. Recursive calls to the algorithm might correspond with looking down a different supermarket aisle.

In summary, if you had to explain to someone else how the QUICKSELECT and PARTITION algorithms work, what pictures would *you* sketch, and what would be *your* play-by-play narrative? What personally-meaningful story or scenario might you use to help yourself and others remember and understand the algorithm's steps? Finally, for what input data sets would you choose to demonstrate your animation? These are the key questions that you'll need to address in this exercise.

**Learning objectives and deliverables.** Keep in mind that, in this part of the study, your main objective is to develop a solid understanding of the QUICKSELECT algorithm, including the PARTITION procedure, works. Upon completing this part of the study, you should be able to

- perform a detailed trace of the QUICKSELECT algorithm (including the PARTITION procedure) operating on any input data set; and
- remember the QUICKSELECT algorithm (including the PARTITION procedure) well enough to implement it in a programming language, if you had to.

To prepare yourself to meet these goals, and to fulfill the requirements of this exercise, please "deliver" the following:

- *Walk through your animation for five input data sets.* Choose at least **five** input data sets on which to instantiate your graphical simulation. Each input data set should include a maximum of **15** data elements, and data element sizes should only range from **1** to **100**. Use the art supplies to construct homemade animations for each of these input data sets. (Note that if you use cut-outs for your data elements, creating five different animations might be as simple as creating some general animation elements that can be used for all input data sets, and then creating five alternative sets of cut-outs—one for each input data sets.) Now carefully walk through your animation, step-by-step, with these input data sets. As you do so, think to yourself (or say aloud, if you'd like) the play-by-play narrative you'd use to explain how the algorithm works.

- *Construct and hand-in a history view of your animation for those input data sets.* The second deliverable actually documents that you have instantiated your animation on five different input data sets. For each of your five input data sets, use an 8½" by 11" sheet of paper to create a "history view" of your animation executing on the input data set. The history view should simply summarize your animation by including "snapshots" of your animation at key points in its execution. For example, if you were animating a sorting algorithm, you might summarize your animation by depicting the state of the array and key variables after each pass of the algorithm's loop; each row in your history view could correspond to one pass of the algorithm's loop.

In sum, think of the homemade animation you'll be developing as an exercise in "implementing" a *general-purpose* graphical simulation. Your animation is general-purpose because you should be able to walk through the animation for any input data set you choose. However, for this exercise, you will only be focusing on a few input data sets, which you should choose carefully so as to cover a wide range of possible cases.

**A word on the story component of the exercise.** Coming up with a story to illustrate any algorithm is challenging! Please don't get hung up for too long trying to come up with a personally-meaningful story that illustrates the algorithm. If, after a few minutes, you still haven't come up with a story. then just go with whatever visual representation of the algorithm comes to mind. If you can come up with a story, then great! But for this exercise, actually constructing a homemade animation, even if it is not based on a story, is more important.

**Taking a break, and asking questions.** Feel free to take a break at any time. (Note that you will be given a half hour lunch break two hours into the session.) If you have questions, raise your hand and I'll come around to help you. If you haven't finished when time is called, please stop what you're doing and hand in what you have finished.

# The QuickSelect Algorithm: Selection in Expected Linear Time[85]

## The Problem

The $i^{th}$ ***order statistic*** of a set of $n$ elements is the $i^{th}$ smallest element. For example, the ***minimum*** of a set of elements is the first order statistic ($i = 1$), and the ***maximum*** is the $n^{th}$ order statistic ($i = n$).

The *selection* problem—that of selecting the $i^{th}$ order statistic from a set of $n$ numbers—can be solved in $O(n \lg n)$ time, since we can sort the numbers using heapsort or merge sort and then simply index the $i^{th}$ element in the output array. There are faster algorithms, however; the algorithm described below solves the selection problem in linear time in the average case.

## The Algorithm

***QUICKSELECT*** is a divide-and-conquer algorithm modeled after the well-known QUICKSORT sorting algorithm. As in QUICKSORT, the idea is to partition the input array recursively. But unlike QuickSort, which recursively processes both sides of the partition, QUICKSELECT only works on one side of the partition. This difference shows up in the analysis: whereas QUICKSORT has an expected running time of $\Theta(n \lg n)$, the expected time of QUICKSELECT is $\Theta(n)$.

### How QuickSelect Works

QUICKSELECT makes use of the same PARTITION procedure used by the QUICKSORT algorithm. (PARTITION is described in detail on the next page.) The following pseudocode for QUICKSELECT returns the $i^{th}$ smallest element in the array $A[left..right]$:

```
QUICKSELECT(A,left,right,i)

1    if left = right
2         then return A[left]
3    mid ← PARTITION(A,left,right)
4    k ← mid – left + 1
5    if i ≤ k
6         then return QUICKSELECT(A,left,mid,i)
7         else return QUICKSELECT(A,mid+1,right,i – k)
8    endif
9    end QUICKSELECT
```

After PARTITION is executed in line 3 of the algorithm, the array $A[left..right]$ is partitioned into two nonempty subarrays $A[left..mid]$ and $A[mid+1..right]$ such that each element of

---

[85]Excerpted and adapted from Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms*. Cambridge, MA: The MIT Press, pp. 153–155, 185, 187–188.

$A[left..mid]$ is less than each element of $A[mid+1..right]$. Line 4 of the algorithm computes the number $k$ of elements in the subarray $A[left..mid]$. The algorithm now determines in which of the two subarrays $A[left..mid]$ and $A[mid+1..right]$ the $i^{th}$ smallest element lies. If $i \leq k$, then the desired element lies on the low side of the partition, and it is recursively selected from the subarray in line 6. If $i > k$, however, then the desired element lies on the high side of the partition. Since we already know $k$ values that are smaller than the $i^{th}$ smallest element of $A[left..right]$—namely, the elements $A[left..mid]$—the desired element is the $(i-k)^{th}$ smallest element of $A[mid+1..right]$, which is found recursively in line 7.

## How PARTITION works

The key to the QUICKSELECT algorithm is the PARTITION procedure, which rearranges the subarray $A[left..right]$ in place. The following pseudocode describes the procedure:

PARTITION(*A*,*left*,*right*)

```
1    piv ← A[left]
2    l ← left
3    r ← right
4    while l < r do
5        while A[l] ≤ piv and l < right do l ← l + 1 endwhile
6        while A[r] > piv and r > left do r ← r − 1 endwhile
7        if (l < r)
8            then exchange A[l] ↔ A[r] endif
9    endwhile
10   mid ← r
11   exchange A[left] ↔ A[mid]
12   return mid
13   end PARTITION
```

Figure 1 shows how PARTITION works. It first selects an element $piv = A[left]$ from $A[left..right]$ as the "pivot" element around which to partition $A[left..right]$. It then grows two regions $A[left..l]$ and $A[r..right]$ from the left and right of $A[left..right]$ respectively, such that every element in $A[left..l]$ is less than or equal to $piv$, and every element in $A[r..right]$ is greater than $piv$.

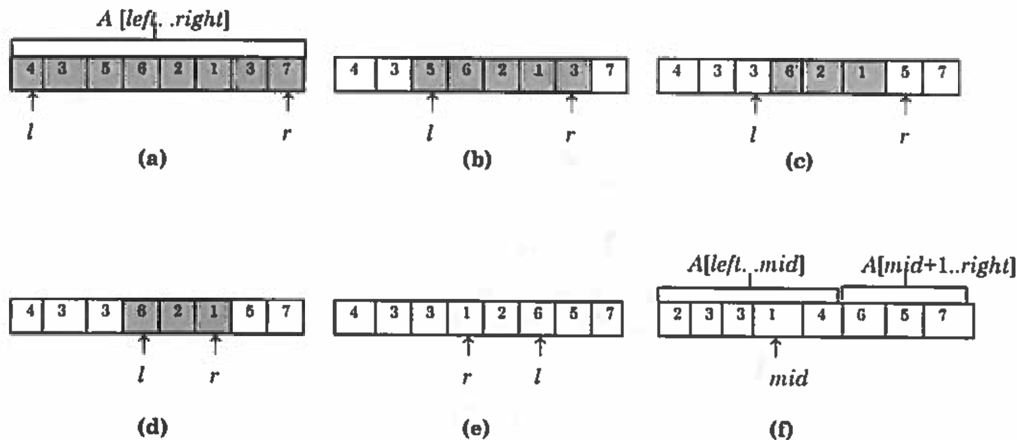



**Figure 1** The operation of PARTITION on a sample array. Unshaded array elements have been placed into the correct partitions, and shaded elements are not yet in their partitions. **(a)** The input array, with the initial values of $l$ and $r$. We partition around *pivot* = $A[l]$ = 4. **(b)** The positions of $l$ and $r$ at line 7 of the first iteration of the main while loop. **(c)** The result of exchanging the elements pointed to by $l$ and $r$ in line 8. **(d)** The positions of $l$ and $r$ at line 7 of the second iteration of the main while loop. **(e)** The positions of $l$ and $r$ at line 7 of the third and last iteration of the main while loop. The loop terminates because now $l \geq r$. **(f)** The final state of $A$, after $A[mid]$ is swapped with $A[left]$ (line 11). The procedure returns the altered array $A$, and $mid = 5$. ). Notice that $A$ has been altered such that array elements up to and including $A[mid]$ are less than or equal to $A[mid]$, and array elements after $A[mid]$ are greater than $A[mid]$.

Within the body of the while loop, the index $l$ is incremented (line 5), and the index $r$ is decremented (line 6), until $A[l] > piv \geq A[r]$. At this point, we know that $A[l]$ is too large to belong to the left-hand region, and $A[r]$ is too small to belong to the right-hand region. Thus, by exchanging $A[l]$ and $A[r]$, we can extend the two regions.

The body of the while loop (lines 5 – 8) repeats until $l \geq r$, at which point the entire array $A[left..right]$ has been partitioned into two subarrays $A[left..r]$ and $A[r+1..right]$ such that all elements in the left-hand partition $A[left..r]$ are less than or equal to pivot, and all elements in the right-hand partition $A[r+1..right]$ are greater than the pivot. As a final step (line 10), the procedure swaps $A[left]$ and $A[mid]$ ($mid = r$; see line 11), so that $A[mid]$ actually contains the pivot element. Thus, at the end of the procedure, we have that $A[left..mid] \leq A[mid] < A[mid+1..right]$. If we were sorting the array $A$ into ascending order, note that the pivot element would now be in its rightful place in the array $A$.

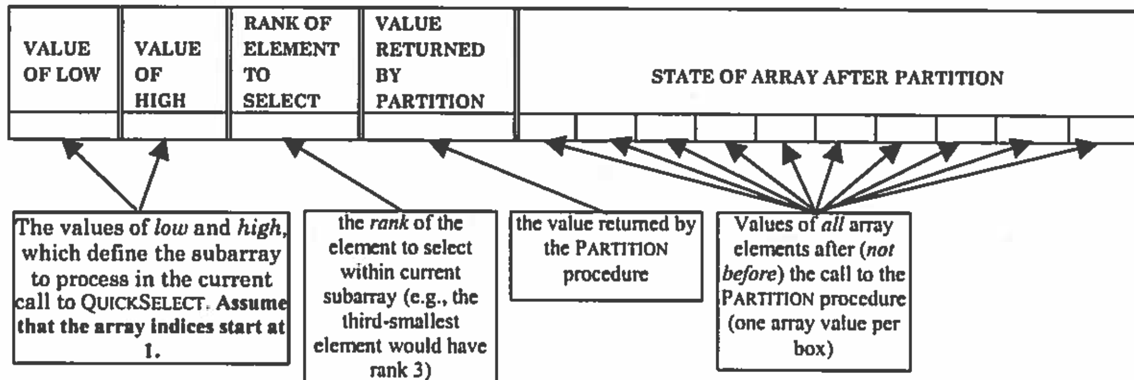
Conceptually, the PARTITION procedure performs a simple function: It puts elements smaller than or equal to the pivot element into the left-hand region of the array, and elements larger than the pivot element into the right-hand region. It then places the pivot element between the two regions, so that all elements to the left of the pivot element are less than or equal to the pivot, and all elements to the right of the pivot element are greater than the pivot. ■

Tracing Task Instructions

## INSTRUCTIONS: TRACING TASKS

**Your Participant Code:________________**

You are about to begin a series of five tracing tasks. For each tracing task, you will be asked to trace through the QuickSelect algorithm for a particular set of input data. Using Microsoft Word, you will document your trace by filling in a predefined table that has the following form (see boxes for explanations of what to fill in for each entry):




Please fill in one row of the table for each recursive call made to the QUICKSELECT algorithm. Be sure to list the *entire* array in each row; every box should have a value. (*Note*: To start a new row within a Microsoft Word table, position the cursor in the last column of the last row of the table, and then hit the tab key.) In addition, please determine the **final output** of the QuickSelect algorithm—the final position of the selected element within the array, as well as its value. Indicate these values by filling in the simple table that appears as follows:

FINAL POSITION OF SELECTED ELEMENT: ________________

VALUE OF SELECTED ELEMENT: ________________

## Before you begin the tracing exercises, please note the following:

1. You will work on a total of **four** tracing exercises. You will have up to **25 minutes** to complete each exercise. Please work as quickly as possible, but do not sacrifice accuracy for speed. Accuracy is more important.

2. As mentioned above, you will be filling in tables directly on the computer. The tables are predefined as *Microsoft Word templates*. When you are instructed begin work on a trace (from within Microsoft Word), select *New...* from the *File* menu. Click on the *Task 2* tab, and then choose the icon labeled

   ***QuickSelect Trace n***

   where ***n*** = 1, 2, 3, or 4. Make sure to choose only the trace number that you are instructed to work on.

3. Once you have created a new document using one of the QuickSelect Trace templates, immediately save the document (*ctrl-s* or *Save* on the *File* menu) in the `C:\Experiment` directory under the following name:

   ***code Trace n***

   where ***code*** is the special code number assigned to you (see the top of this document for your code), and ***n*** is the number of the trace you are about to perform (1 – 4). For example, "CN#23 Trace 3" might be a valid filename. *Because the file timestamps will be used to calculate how much time you spend on each exercise, it is essential that you* ***save*** *the document before doing anything else.*

4. Please remember to save your document periodically as you work.

5. If you finish a tracing exercise before time is called, first save it (*ctrl-s* or *Save* on the *File* menu), and then close it (*Close* on the *File* menu). Finally, turn your monitor off, to indicate that you are done. ***Do not go on to another exercise until you are instructed to do so.***

If you have questions at any point, please raise your hand, and I will come around to help you.

Tracing Tasks

# QUICKSELECT TRACING EXERCISE #1

Please trace through the execution of the QuickSelect algorithm for the following input data:

Select the 4th smallest element from [12,6,9,24,3,12,4,11,15,8]

Refer to the handout labeled "Tracing Exercise Instructions" for detailed instructions, and for detailed explanations of the trace values to fill in below.

| VALUE OF LOW | VALUE OF HIGH | RANK OF ELEMENT TO SELECT | VALUE RETURNED BY PARTITION | STATE OF ARRAY AFTER PARTITION | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 4 | | | | | | | | | | | |
| | | | | | | | | | | | | | |

FINAL POSITION OF SELECTED ELEMENT: ____________________

VALUE OF SELECTED ELEMENT: ____________________

# QUICKSELECT TRACING EXERCISE #2

Please trace through the execution of the QuickSelect algorithm for the following input data:

**Select the smallest element from [7,20,11,13,19,8,14,22,17,16]**

Refer to the handout labeled "Tracing Exercise Instructions" for detailed instructions, and for detailed explanations of the trace values to fill in below.

| VALUE OF LOW | VALUE OF HIGH | RANK OF ELEMENT TO SELECT | VALUE RETURNED BY PARTITION | STATE OF ARRAY AFTER PARTITION | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 1 | | | | | | | | | | | |
| | | | | | | | | | | | | | |

**FINAL POSITION OF SELECTED ELEMENT:** __________

**VALUE OF SELECTED ELEMENT:** __________

# QUICKSELECT TRACING EXERCISE #3

Please trace through the execution of the QuickSelect algorithm for the following input data:

**Select the 7th smallest element from [11,3,19,8,20,15,22,9,19,13]**

Refer to the handout labeled "Tracing Exercise Instructions" for detailed instructions, and for detailed explanations of the trace values to fill in below.

| VALUE OF LOW | VALUE OF HIGH | RANK OF ELEMENT TO SELECT | VALUE RETURNED BY PARTITION | STATE OF ARRAY AFTER PARTITION | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 7 | | | | | | | | | | | |
| | | | | | | | | | | | | | |

**FINAL POSITION OF SELECTED ELEMENT:** ____________________

**VALUE OF SELECTED ELEMENT:** ____________________

# QUICKSELECT TRACING EXERCISE #4

Please trace through the execution of the QuickSelect algorithm for the following input data:

**Select the 5th smallest element from [27,23,20,19,16,15,12,9,7,3]**

Refer to the handout labeled "Tracing Exercise Instructions" for detailed instructions, and for detailed explanations of the trace values to fill in below.

| VALUE OF LOW | VALUE OF HIGH | RANK OF ELEMENT TO SELECT | VALUE RETURNED BY PARTITION | STATE OF ARRAY AFTER PARTITION | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 5 | | | | | | | | | | | |
| | | | | | | | | | | | | | |

FINAL POSITION OF SELECTED ELEMENT: ________________

VALUE OF SELECTED ELEMENT: ________________

Programming Task Instructions

# INSTRUCTIONS: PROGRAMMING TASK

**Your Participant Code:**

**Overview.** In the task you're about to begin, you'll implement, in a real programming language, the QUICKSELECT algorithm that you previously studied. Fortunately, you don't have to write your implementation from scratch. Instead, you'll work with a *skeleton program* written in either Java or C++ (your choice). The skeleton program

- defines a `QSelectAlg` class containing all necessary data and methods;
- implements for you a method that requests input data from the user and initializes class data members appropriately;
- interfaces the class with a main program that reports the algorithm's output to the user; and
- stubs out the particular code segments that you are responsible for implementing.

The skeleton program thus provides a solid starting point, enabling you able to focus exclusively on implementing the QUICKSELECT algorithm. Note that you will have 35 minutes to work on this exercise.

**Steps.** To complete the implementation task, follow these steps:

1. *Choose a language.* Attached you'll find a complete printout of the skeleton program in both Java and C++. You might want to take a look at both printouts in order to help you choose a language.

2. *Open the skeleton program for editing.* Once you've chosen a language, open (or maximize) Microsoft Word, and choose *New...* from the *File* menu. Click on the *Task 2* tab, and then double-click on the icon corresponding to the programming language you've chosen—*Java Skeleton* if you'd like to program in Java, or *C++ Skeleton* if you'd like to program in C++. The corresponding skeleton program will then come up in a new document. *Immediately save the document by typing* ctrl-s *or by choosing* Save *from the* File *menu.* Save the document in the `C:\Experiment` directory under the following name:

   *code* Program *lang*

   where *code* corresponds to your participant code (see top of this page), and *lang* corresponds to the programming language you've chosen (*C++* or *Java*). For example, "NC53 Program Java" might be a valid file name. *Because the file timestamps will be used to calculate how much time you spend on each exercise, it is essential that you save the document before doing anything else.*

3. *Implement the missing code.* You'll notice that the skeleton code is color-coded to enhance its readability. In particular, comments are shaded green, keywords are shaded

blue, and (most important for you) the places where you must fill in your own code are clearly marked with red `***TO DO***` blocks containing brief instructional message. Your task is to complete the implementation of the algorithm by filling in the missing code. In essence, you will need to implement three pieces of code:

- The recursive `select()` method;
- The `partition()` method; and
- The parameter list in the original call to the `select` method, which is located within the `QSelect()` method.

Filling in this code will round out a complete implementation of the version of the QuickSelect algorithm that you have been exploring in this study. (Note: *As you implement code, please do not worry about color-coding your code according to the conventions established in the file. Also remember to save the file periodically as you work.*

4. *Finishing up the task.* Work on the task until you are satisfied that your program is free of syntactic and semantic errors, or until time is called. If you complete the task before time is called, please close your file, making sure to save it first. Then turn your monitor off in order to indicate that your are done.

## Java QuickSelect Skeleton

```
/*******************************************************************
                        QSELECTALG.JAVA

       Your Participant Code:_______________
=====================================================================
This file contains the source code for the QSelect class, a Java
implementation of the QuickSelect algorithm you've been learning about
in this study.

Some of the class has already been implemented for you—most significantly,
the GetInputData method, which obtains the input data from the user and
initializes the member data (the array a, the rank kth, and the size of
the array num_elts).

Your task is to complete the implementation of the class by filling in
the missing code.  All places where you are to fill in code are clearly
marked with commented red ***TO DO*** blocks.  In essence, you will be
implementing two methods--the recursive select procedure and the
partition procedure.  These methods will round out a complete
implementation of the version of the QuickSelect algorithm that you have
been exploring in this study.
*******************************************************************/

import java.io.*;

public class QSelectAlg
{
   private static DataInputStream input = new DataInputStream (System.in);

   private final int MAX_SELECT_ELTS = 50; // Maximum number of elements to select from
   private final int MIN_VALUE = 1;        // Minimum value any element can have
   private final int MAX_VALUE = 100;      // Maximum value any element can have

   private int a []; //The array containing the integers to select from; allocated within
                  //GetInputData().

   private int num_elts; //The number of elements in a;
                     //initialized within GetInputData()

   private int kth;  //The rank of the element to select;
                  //initialized within GetInputData()


   public static void main (String argv [])
   //*******************************************************************
   // The main program executes a do-while loop in which the GetInputData() method
   // is called to initialize the input data; the QSelect() method is called to
   // execute the algorithm on those input data; and the user is given the option
   // of running the algorithm on a new set of input
   //*******************************************************************

   {
          QSelectAlg Q = new QSelectAlg();
          try
          {
                 char ans;
                 do {
                        Q.GetInputData();
                        int result = Q.QSelect();
                        System.out.println ("The QuickSelect algorithm returned " + result + ".");
                        System.out.println ("Would you like to run the algorithm on a new input data set
                                             (Y/N)?");
                        ans = input.readLine ().charAt (0);
                 } while (ans == 'y' || ans == 'Y');
          }
          catch (IOException e)
          {
                 // Just end the program
          };
   }

public QSelectAlg()

   //*******************************************************************
   // GIVEN  : Nothing
   // TASK   : Perform preliminary initializations of member data
   //*******************************************************************
   {
      num_elts = 0;
      kth = 0;
   }
```

```
public void GetInputData()
   //*******************************************************************
   // GIVEN   : Nothing
   // TASK    : Request from the user the input data for an execution of the
   //           the QuickSelect algorithm.  In particular, ask the user to
   //           input the size of the array, the array elements, and the rank
   //           of the element to select.  Perform minimal error-checking to
   //           ensure that the input data are valid.
   //*******************************************************************
   {
      System.out.println ("***QUICKSELECT ANIMATION***");
      System.out.print ("Please enter number of elements to be selected from (" + 2 + " - " +
                        MAX_SELECT_ELTS + "):");
      num_elts = getInt (2, MAX_SELECT_ELTS);
      a = new int[num_elts];
      System.out.println ("Enter the integer elements, one per line");
      for (int count=0; count<num_elts; ++count)
      {
             a [count] = getInt (MIN_VALUE, MAX_VALUE);
      }
      System.out.print ("Enter the rank of the element you wish to select:");
      kth = getInt (1, num_elts);
   }


   public int QSelect()
   //*******************************************************************
   // GIVEN   : Nothing
   // TASK    : Execute the QuickSelect algorithm on the input data that have
   //           been entered. (You MUST call the GetData() method before
   //           calling this method.)
   // RETURN  : The value of the kth smallest element in a.
   //*******************************************************************
   {
      return select( /* ***TO DO*** FILL IN PROPER PARAMETERS HERE */ );
   }

   private int partition( /* ***TO DO***  FILL IN PROPER PARAMETERS HERE */ )
   //*******************************************************************
   // GIVEN   : ??? (***TO DO***  FILL IN DOCUMENTATION HERE)
   // TASK    : Partition the current subarray a[j..k] into two sections
   //           a[j..m] and a[m+1..k], such that all elements a[j..m] are
   //           less than or equal to a[m], and all elements a[m+1..k] are
   //           greater than m.
   // METHOD : ALWAYS CHOOSE A[J] (THE FIRST ELEMENT IN THE CURRENT SUBARRAY)
   //           AS THE PIVOT ELEMENT
   // RETURN : m, the location of the pivot element after the partitioning
   //           is finished.
   //******************************************************************

   {
      /* ***TO DO*** FILL IN THE CODE FOR THE PARTITION METHOD HERE */
   }


   private int select( /* ***TO DO***  FILL IN PROPER PARAMETERS HERE */ )

   //*******************************************************************
   // GIVEN   : ??? (***TO DO***  FILL IN DOCUMENTATION HERE)
   // TASK    : select the kth element from the current subarray
   // METHOD : Use the recursive divide-and-conquer QuickSelect algorithm
   //          that you have been exploring in this study; call on the
   //          paritition routine (which you implemented above) as needed.
   // RETURN:   The value of the kth smallest element in a, (remember that
   //          kth and a are member data)
   //*******************************************************************

   {
      /* ***TO DO*** FILL IN THE CODE FOR THE SELECT METHOD HERE */
   }
```

```
private int getInt (int low, int high)
//*******************************************************************
// GIVEN  : low and high, defining the range of valid input data
// TASK   : Read in an integer value, making sure that is within range
// RETURN : The valid integer value that was read in
//*******************************************************************

{
          int result = low;
          boolean success;
          do
          {
                   success = false;
                   try
                   {
                             result = Integer.parseInt (input.readLine ());
                             if (result < low || result > high)
                                       throw new IOException ("value out of range");
                             success = true;
                   }
                   catch (Exception e)
                   {
                System.out.println ("You must enter an integer between " + low + " and " +
                                     high + ".");
                   }

          } while ( ! success);
          return result;
 }
}
```

## C++ QuickSelect Skeleton

```
/*********************************************************************
                        QSELECT.CPP

       Your Participant Code:________________
======================================================================
This file contains the source code for the QSelect class, a C++
implementation of the QuickSelect algorithm you've been learning about
in this study.

Some of the class has already been implemented for you--in particular,
the class constructor and destructor, as well as the GetInputData method,
which obtains the input data from the user and initializes the member data
(the array a, the rank kth, and the size of the array num_elts).

Your task is to complete the implementation of the class by filling in
the missing code.  All places where you are to fill in code are clearly
marked with red ***TO DO*** blocks.  In essence, you will be
implementing two methods--the recursive select procedure and the
partition procedure.  These methods will round out a complete
implementation of the version of the QuickSelect algorithm that you have
been exploring in this study.
*********************************************************************/

#include <stdio.h>
const int max_select_elts = 50; // Maximum number of elements to select from
const int min_value = 1;        // Minimum value any element can have
const int max_value = 100;      // Maximum value any element can have

class QSelectAlg {

public:
   QSelectAlg() {
   //*********************************************************************
   // GIVEN  : Nothing
   // TASK   : Perform preliminary initializations of member data
   //*********************************************************************
      num_elts = 0;
      kth = 0;
      a = NULL;
   }

   void GetInputData()
   //*********************************************************************
   // GIVEN  : Nothing
   // TASK   : Request from the user the input data for an execution of the
   //          the QuickSelect algorithm.  In particular, ask the user to
   //          input the size of the array, the array elements, and the rank
   //          of the element to select.  Perform minimal error-checking to
   //          ensure that the input data are valid.
   // METHOD : Helper method getInt accepts the actual input
   // RETURN : Nothing, but num_elts, a, and kth are properly initialized
   //*********************************************************************
   {
      int count;
      printf("***QUICKSELECT ANIMATION***\n");
      printf("Please enter number of elements to be selected from (2 - %d):\n",
              max_select_elts);
      num_elts = getInt(2,max_select_elts);
      if (a != NULL) //We need to free up a's heap space before re-allocating a
         delete [] a;
      a = new int[num_elts];
      printf("Enter the integer elements, one per line\n");
      for (count = 0; count < num_elts; ++count)
         a[count] = getInt(min_value,max_value);
      printf("Please enter the rank of the element to select (1 - %d)\n",num_elts);
      kth = getInt(1,num_elts);
  }


   int QSelect()
   //*******************************************************************
   // GIVEN  : Nothing
   // TASK   : Execute the QuickSelect algorithm on the input data that have
   //           been entered. (You MUST call the GetData() method before
   //           calling this method.)
   // RETURN:   The value of the kth smallest element in a.
   //*******************************************************************
   {
      return select(/* ***TO DO*** FILL IN PROPER PARAMETERS HERE */);
   }
```

```
   ~QSelectAlg() {
   //*********************************************************************
   // GIVEN  : Nothing
   // TASK   : Free the array heap space.
   //********************************************************************
      delete [] a;
   }

private:
   int *a; //The array containing the integers to select from;
           //dynamically allocated on the heap within GetInputData().

   int num_elts; //The number of elements in a;
                 //initialized within GetInputData()

   int kth; //The rank of the element to select;
            //initialized within GetInputData()

   int getInt(int low, int high)
   //*********************************************************************
   // GIVEN  : low and high, defining the range of valid input data
   // TASK   : Read in an integer value, making sure that it is within range
   // RETURN : The valid integer value that was read in
   //*********************************************************************

   {
       int success, result;
          do {
             success = scanf("%d", &result);
             if ((!success) || (result < low) || (result > high))
                printf("You must enter an integer between %d and %d. Please reenter.\n",
                low, high);
          } while ((!success) || (result < low) || (result > high));
       return result;
   }

   int partition( /* ***TO DO***  FILL IN PROPER PARAMETERS HERE */ )
   //*********************************************************************
   // GIVEN  : ??? (***TO DO***:  FILL IN DOCUMENTATION HERE)
   // TASK   : Partition the current subarray a[j..k] into two sections
   //          a[j..m] and a[m+1..k], such that all elements a[j..m] are
   //          less than or equal to a[m], and all elements a[m+1..k] are
   //          greater than m.
   // METHOD : ALWAYS CHOOSE A[J] (THE FIRST ELEMENT IN THE CURRENT SUBARRAY)
   //          AS THE PIVOT ELEMENT
   // RETURN:  m, the location of the pivot element after the partitioning
   //          is finished.
   //*********************************************************************

   {
      /* ***TO DO*** FILL IN THE CODE FOR THE PARTITION METHOD HERE */
   }


   int select( /* ***TO DO***  FILL IN PROPER PARAMETERS HERE */ )

   //********************************************************************
   // GIVEN  : ??? (***TO DO***:  FILL IN DOCUMENTATION HERE)
   // TASK   : select the kth element from the current subarray
   // METHOD : Use the recursive divide-and-conquer QuickSelect algorithm
   //         that you have been exploring in this study; call on the
   //         paritition routine (which you implemented above) as needed.
   // RETURN:   The value of the kth smallest element in a, (remember that
   //         kth and a are member data)
   //******************************************************************

   {
      /* ***TO DO*** FILL IN THE CODE FOR THE SELECT METHOD HERE */
   }
};
```

```
void main()
//******************************************************************
// The main program executes a do-while loop in which the GetInputData() method
// is called to initialize the input data; the QSelect() method is called to
// execute the algorithm on those input data; and the user is given the option
// of running the algorithm again on a new set of input data.
//******************************************************************

{
   char ch;
   int result;
   QSelectAlg *Q = new QSelectAlg();
   do {
      Q->GetInputData();
      result = Q->QSelect();
      printf("The QuickSelect algorithm returned %d.\n",result);
      printf("Would you like to run the algorithm on a new input data set (Y/N)?\n");
      scanf("%1s", &ch);
   } while ((ch == 'Y') || (ch == 'y'));
   delete Q;
}
```

Instructions for Finishing Up

## INSTRUCTIONS: FINISHING UP THE STUDY

**Congratulations!** you are just about done with this study. **Thank you** so much for agreeing to participate. You can leave feeling good that your participation not only has helped me to complete my dissertation research, but also has contributed to the advancement of science! Before you leave and receive your check for participating, please do the following:

1. *Back up your files to floppy disk.* I have given you a floppy disk labeled with your participant code. Please use *My Computer* or *Windows Explorer* in Microsoft Windows to ***copy*** (not *move*!) all of your files from the `C:\Experiment` directory to that floppy disk (`A:\`). If you are unfamiliar with Microsoft Windows and need help copying the files, raise your hand and I'll come around to help you. When you are done copying the files, please remove the floppy disk from the disk drive and place it with your other materials.

2. *Fill out the exit questionnaire.* Attached you'll find a brief exit questionnaire that asks you to assess your experiences in this study. Please take a few minutes to provide written responses to the five questions on the questionnaire. If you need more space, feel free to use the back of the questionnaire or additional sheets of scratch paper.

When you have completed these two things, raise your hand and I'll come around to give you your check. Thanks again!

Exit Questionnaire: Active Viewing Group

## ALGORITHM ANIMATION STUDY: EXIT QUESTIONNAIRE (AV)

*(Feel free to use the back of this sheet if you need more space.)*

1. Do you feel that studying and interacting with the algorithm animation helped you to learn the QuickSelect algorithm? Why or why not?

2. What, if anything, did you like about using algorithm animation to learn about how the QuickSelect algorithm works? What, if anything, did you not like?

3. Do you feel you had sufficient time to study and interact with the QuickSelect animation? If not, how much more time would you have liked? What would you have done in the extra time?

4. Do you feel you had sufficient time to complete the tracing and programming exercises? If not, how much more time do you feel you would have needed?

5. If you had known in advance the exact exercises (i.e., the tracing and the programming exercises) that you would be tested on , would you have done anything differently during the learning phase of this study? If so, what?

6. In the future, would you use algorithm animation as a study or learning aid? If so, In what capacity?

Exit Questionnaire: Self-Construction Group

## ALGORITHM ANIMATION STUDY: EXIT QUESTIONNAIRE (SC)

*(Feel free to use the back of this sheet if you need more space.)*

1. Do you feel that the homemade algorithm animation construction exercise helped you to learn how the QUICKSELECT algorithm works? Why or why not?

2. What, if anything, did you like about constructing your own animation as a means of learning about how the QUICKSELECT algorithm works? What, if anything, did you not like?

3. Do you feel that you had sufficient time to construct your animation, simulate it on input data sets, and construct history views? If not, how much more time would you have needed? What would you have done in that extra time?

4. Do you feel you had sufficient time to complete the tracing and programming exercises? If not, how much more time do you feel you would have needed?

5. If you had known in advance the exact exercises (i.e., the tracing and the programming exercises) that you would be tested on , would you have done anything differently during the animation construction phase of this study? If so, what?

6. In the future, would you use algorithm animation construction as a study or learning aid? If so, In what capacity?

# APPENDIX E

# EXPERIMENT DATA

EXPERIMENT DATA

| P# | GPA | Treatment | Trace #1 Score | (Perfect: Pct | 40) Time | Trace #2 Score | (Perfect: Pct | 8) Time | Trace #3 Score | (Perfect: Pct | 32) Time | Trace #4 Score | (Perfect: Pct | 63) Time | Trace Totals Score | (Perfect: Pct | 143) Time | Program Score | (Perfect: Pct | 31) Time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 3.60 | AVP | 15.17 | 37.93% | 24.32 | 5.00 | 62.50% | 4.60 | 12.53 | 39.16% | 9.75 | 8.27 | 13.13% | 13.97 | 41 | 28.65% | 52.63 | 3.00 | 9.68% | 29.45 |
| 16 | 3.20 | AVP | 33.00 | 82.50% | 21.92 | 7.00 | 87.50% | 3.08 | 20.80 | 65.00% | 18.27 | 62.00 | 98.41% | 19.90 | 123 | 85.87% | 63.17 | 25.00 | 80.65% | 33.05 |
| 18 | 2.78 | AVP | 22.12 | 55.30% | 16.50 | 7.00 | 87.50% | 2.38 | 22.00 | 68.75% | 20.27 | 62.00 | 98.41% | 14.80 | 113.12 | 79.10% | 53.95 | 15.50 | 50.00% | 35.05 |
| 20 | 2.48 | AVP | 1.67 | 4.18% | 25.25 | 5.00 | 62.50% | | 4.47 | 13.97% | 24.25 | 20.80 | 33.02% | 13.53 | 31.94 | 22.34% | 63.03 | 4.50 | 14.52% | 35.48 |
| 22 | 2.23 | AVP | 19.30 | 48.25% | 12.95 | 4.00 | 50.00% | 2.13 | 13.60 | 42.50% | 11.05 | 49.00 | 77.78% | 10.28 | 85.90 | 60.07% | 36.42 | 10.50 | 33.87% | 34.98 |
| 24 | 2.51 | AVP | 10.09 | 25.23% | 19.90 | 3.00 | 37.50% | 4.75 | 6.07 | 18.97% | 18.58 | 33.30 | 52.86% | 19.23 | 52.46 | 36.69% | 62.47 | 2.50 | 8.06% | 39.63 |
| 2 | 4.01 | AVT | 32.55 | 81.38% | 9.30 | 8.00 | 100.00% | 1.58 | 29.27 | 91.47% | 7.27 | 47.00 | 74.60% | 7.72 | 116.82 | 81.69% | 25.87 | 24.00 | 77.42% | 25.43 |
| 4 | 3.20 | AVT | 34.00 | 85.00% | 18.10 | 7.00 | 87.50% | 2.08 | 32.00 | 100.00% | 17.05 | 62.00 | 98.41% | 14.88 | 135.00 | 94.41% | 52.12 | 22.00 | 70.97% | 35.32 |
| 6 | 3.04 | AVT | 33.86 | 84.65% | 17.15 | 8.00 | 100.00% | 2.75 | 28.00 | 87.50% | 12.22 | 54.00 | 85.71% | 11.33 | 123.86 | 86.62% | 43.45 | 27.50 | 88.71% | 27.25 |
| 8 | 3.58 | AVT | 19.24 | 48.10% | 12.07 | 6.00 | 75.00% | 1.90 | 28.00 | 87.50% | 8.37 | 33.00 | 52.38% | 11.00 | 86.24 | 60.31% | 33.33 | 22.50 | 72.58% | 28.30 |
| 10 | 3.19 | AVT | 26.00 | 65.00% | 25.07 | 7.00 | 87.50% | 4.23 | 9.20 | 28.75% | 19.50 | 44.00 | 69.84% | 15.33 | 86.20 | 60.28% | 64.13 | 27.00 | 87.10% | 31.25 |
| 12 | 3.07 | AVT | 21.12 | 52.80% | 20.12 | 6.00 | 75.00% | 3.25 | 9.53 | 29.78% | 21.10 | 12.21 | 19.38% | 21.67 | 48.86 | 34.17% | 66.13 | 3.00 | 9.68% | 35.13 |
| 13 | 3.53 | SCP | 33.00 | 82.50% | 11.80 | 8.00 | 100.00% | 2.60 | 20.80 | 65.00% | 11.75 | 63.00 | 100.00% | 12.83 | 124.80 | 87.27% | 38.98 | 23.17 | 74.74% | 27.12 |
| 15 | 3.12 | SCP | 38.00 | 95.00% | 16.50 | 8.00 | 100.00% | 1.27 | 23.80 | 74.38% | 9.32 | 59.00 | 93.65% | 17.57 | 128.80 | 90.07% | 44.65 | 27.50 | 88.71% | 32.45 |
| 17 | 2.97 | SCP | 35.00 | 87.50% | 25.70 | 8.00 | 100.00% | 2.73 | 21.70 | 67.81% | 15.52 | 26.65 | 42.30% | 24.20 | 91.35 | 63.88% | 68.15 | 28.50 | 91.94% | 20.47 |
| 19 | 2.70 | SCP | 35.00 | 87.50% | 13.83 | 7.00 | 87.50% | 2.98 | 18.80 | 58.75% | 5.30 | 49.00 | 77.78% | 10.83 | 109.80 | 76.78% | 32.95 | 23.50 | 75.81% | 33.73 |
| 23 | 2.30 | SCP | 7.89 | 19.73% | 25.98 | 4.00 | 50.00% | 4.17 | 4.20 | 13.13% | 25.33 | 16.57 | 26.30% | 25.12 | 32.66 | 22.84% | 80.60 | 30.00 | 96.77% | 25.08 |
| 1 | 4.01 | SCT | 23.00 | 57.50% | 14.20 | 6.00 | 75.00% | 2.75 | 20.80 | 65.00% | 13.13 | 21.90 | 34.76% | 8.05 | 71.70 | 50.14% | 38.13 | 12.75 | 41.13% | 32.72 |
| 3 | 3.83 | SCT | 35.00 | 87.50% | 14.58 | 7.00 | 87.50% | 1.48 | 23.80 | 74.38% | 10.47 | 60.00 | 95.24% | 11.93 | 125.80 | 87.97% | 38.47 | 26.67 | 86.03% | 22.23 |
| 5 | 3.38 | SCT | 27.63 | 69.08% | 24.63 | 8.00 | 100.00% | 8.53 | 16.03 | 50.09% | 25.77 | 26.80 | 42.54% | 24.48 | 78.46 | 54.87% | 83.42 | 10.00 | 32.26% | 35.23 |
| 7 | 3.19 | SCT | 22.00 | 55.00% | 21.18 | 8.00 | 100.00% | 4.05 | 11.10 | 34.69% | 17.20 | 17.60 | 27.94% | 9.53 | 58.70 | 41.05% | 51.97 | 5.50 | 17.74% | 33.42 |
| 9 | 3.07 | SCT | 22.55 | 56.38% | 25.28 | 6.00 | 75.00% | 8.90 | 17.80 | 55.63% | 26.07 | 41.85 | 66.43% | 17.27 | 88.20 | 61.68% | 77.52 | 14.50 | 46.77% | 35.42 |
| 11 | 2.66 | SCT | 32.17 | 80.43% | 14.47 | 8.00 | 100.00% | 4.68 | 17.27 | 53.97% | 11.10 | 21.97 | 34.87% | 7.38 | 79.41 | 55.53% | 37.63 | 15.50 | 50.00% | 35.60 |
| Mean | | (AV): | 22.34 | 55.86% | 18.56 | 6.08 | 76.04% | 2.98 | 17.96 | 56.11% | 15.64 | 40.63 | 64.49% | 14.47 | 87.01 | 60.85% | 51.39 | 15.58 | 50.27% | 32.53 |
| Mean | | (SC): | 28.29 | 70.74% | 18.92 | 7.09 | 88.64% | 4.01 | 17.83 | 55.71% | 15.54 | 36.76 | 58.35% | 15.38 | 89.97 | 62.92% | 53.86 | 19.78 | 63.81% | 30.32 |
| Std. Dev. | | (AV): | 10.71 | 26.78% | 5.47 | 1.64 | 20.50% | 1.17 | 9.97 | 31.14% | 5.62 | 17.75 | 28.17% | 3.67 | 35.68 | 24.95% | 13.46 | 9.92 | 32.01% | 4.32 |
| Std. Dev. | | (SC): | 8.99 | 22.49% | 5.44 | 1.30 | 16.25% | 2.56 | 6.34 | 19.83% | 7.25 | 19.21 | 30.49% | 6.42 | 32.60 | 22.91% | 16.88 | 9.32 | 31.56% | 5.06 |

# APPENDIX F

# SALSA LANGUAGE SUMMARY

# Commands

## Storyboard Element Creation, Deletion, and Attribute Modification

| COMMAND: | **Create** |
|---|---|
| SYNTAX: | (1) **create** <element> <name> [<initial-attribute-settings>]<br><br>(2) **create** <element> <name> **as clone of** <extant-element> [<overridden attributes>]<br><br>(3) **create refpoint** <name> **of** <cutout-ref> **at** <pos> |
| DESCRIPTION: | <name> identifies a new <element> (cutout, position, or grid) to be instantiated. <initial-attribute-settings> is an optional list of attribute-value pairs of the following form: -<attribute-name> <value>. For each such pair, <value> is assigned to the corresponding attribute identified by <attribute-name>. See the tables below for a complete list of the attributes associated with cutouts and grids.<br><br>A new <element> (cutout, position, or grid) called <name> is created, taking on the appearance of the prototype <extant-entity>. <name> assumes all of the attribute values of <extant-entity>, except for those explicitly overridden in <overridden-attributes>, which is a list of attribute-value pairs as above.<br><br><name> identifies a new refpoint to be associated with the existing cutout <cutout-ref>; the position <pos> of the refpoint is in relative Cartesian coordinates, but in practice is specified by direct manipulation. The refpoint <name> may be used like any other attribute of <cutout-ref>. |
| EXAMPLES: | create cutout foo -visible false<br>create cutout bar as clone of foo -visible true<br>create refpoint arm of foo at 0.6, 0.7 |

| COMMAND: | **Place** |
|---|---|
| SYNTAX: | **place** <name> **at** <location>; |
| DESCRIPTION: | Positions the (previously-created) storyboard element <name> at <location>, which is either specified by direct-manipulation (e.g., you simply drag the storyboard element into the storyboard window and drop it where you want), or is specified in terms of its relationship to storyboard elements that are already in the storyboard. |
| EXAMPLES: | place player at gridpos 2,2 of mygrid<br>place bottom-center of arrow at top-border of gridpos 2,2 of mygrid |

| COMMAND: | **Delete** |
|---|---|
| SYNTAX: | **delete** <entity-ref> |
| DESCRIPTION: | Destroys <entity-ref>; the entity disappears from the screen and cannot be reclaimed. |
| EXAMPLES: | delete mygrid<br>delete mycutout |

| COMMAND: | **Assign** |
|---|---|
| SYNTAX: | **assign** <alternative-name> **to** <entity-ref> |
| DESCRIPTION: | Gives extant storyboard element <entity-ref> an alternative name <alternative-name>. |
| EXAMPLES: | assign current_max to cutout in a left-of cutout at gridpos 1 of a |

| COMMAND: | **Set** |
|---|---|
| SYNTAX: | **set** <attribute> **of** <entity-ref> **to** <value> |
| DESCRIPTION: | Explicitly sets the attribute <attribute> of the storyboard element <entity-ref> to the new value <value>. See the tables below for a complete listing of attributes. |
| EXAMPLES: | set number of rows of my-grid to 5;<br>set highlighted of my-cutout to true; |

## Conditionals and Iteration

| COMMAND: | **If-then-else** |
|---|---|
| SYNTAX: | **if** <boolean-test> then<br>  <SALSA-command-list><br>**else**<br>  <SALSA-command-list><br>**endif** |
| DESCRIPTION: | Supports conditional branching. <boolean test> can take one of two forms:<br>• *A spatial test.* Uses one of the 10 spatial relations keywords (p. 2) to compare the spatiality of two storyboard elements—for example, *my-cutout is left-of your-cutout.*<br>• *An attribute test.* Tests or compares the non-spatial attributes of storyboard elements—for example, *highlighted of my-cutout is true; height of my-cutout < height of your-cutout.* |
| EXAMPLES: | if my-cutout is below your-cutout<br>  move mycutout up 2.5 * height of my-cutout<br>else<br>  flash yourcutout for 1.5 sec<br>endif |

| COMMAND: | **While** |
|---|---|
| SYNTAX: | **while** <boolean-test> do<br>  <SALSA-command-list><br>**endwhile** |
| DESCRIPTION: | Supports conditional iteration. See description of If-the-else for an explanation of <boolean-test>. |
| EXAMPLES: | while mycutout is left-of left-center of mygrid<br>  move cutout right cellwidth of mygrid<br>  flash cutout for 1.5 sec;<br>  flash cutout in mygrid above mycutout out for 1.5 sec;<br>endwhile |

| COMMAND: | **Foreach** |
|---|---|
| SYNTAX: | (1) **foreach** gridpos <pos> **of** <entity-ref> [**from** <row>,<col> **to** <row>,<col>] do<br>  <SALSA-command-list><br>**endforeach**<br><br>(2) **foreach** <entity> <iterator> [**such that** <boolean test involving iterator>]<br>  <SALSA-command-list><br>**endforeach** |
| DESCRIPTION: | Supports iteration over a set of storyboard elements (*cutouts, positions*, or *grid positions*):<br><br>(1) Iterates over a set of positions associated with a grid; the optional *from-to* clause allows a specific range of grid positions to be specified.<br><br>(2) Iterates over all storyboard elements that satisfy a certain boolean test (see the description of *if-then-else* above for an explanation of boolean tests in SALSA). |
| EXAMPLES: | foreach cutout c such that c is left-of mycutout do<br>  move c right 1.5 * width of mycutout<br>endforeach<br><br>foreach gridpos p of mygrid from 1,1 and 7,1<br>  if p is unoccupied then<br>    move mycutout to pos<br>  endif<br>endforeach |

## Animation

| COMMAND: | **Move** |
|---|---|
| SYNTAX: | `(1) move <entity-ref> to <position-ref> [over <time> sec] [along <path-spec> path]`<br>`(2) move <entity-ref> <direction> <distance> [over <time> seconds] [along <path-spec> path]` |
| DESCRIPTION: | Causes a cutout or grid to move from its present location to a new location:<br>(1) Requires a position (possibly relative to the location of an extant object) to be specified as the destination of the move,<br>(2) Uses a distance (as a percentage of the height or width of an extant storyboard object) and direction (left, right, up, down) to determine the destination of the move.<br>(3) Both (1) and (2) take optional information regarding the duration of the move (in seconds and fractions thereof, and the shape of the path (*straight, jump, clockwise, counterclockwise, custom*). In the case of a custom path, the user specifies the path via direct manipulation. |
| EXAMPLES: | `move mycutout to top-center of arrow over 2.0 sec along counterclockwise path`<br>`move mycutout down (1.5 * width of mycutout)` |

| COMMAND: | **Resize** |
|---|---|
| SYNTAX: | `resize <cutout-ref> to <height-factor>, <width-factor> [over <time> sec]` |
| DESCRIPTION: | Changes the size of cutout `<cutout-ref>`. The new size of `<cutout-ref>` is specified by `<height-factor>` and `<width-factor>`, both of which must be expressions involving the heights and widths of existing objects. The duration of the resizing can be specified using the optional *over* `<time>` *sec* clause. |
| EXAMPLE: | `resize mycutout to 0.5 * height of mycutout, 0.3 * width of mycutout over 1.0 sec` |

| COMMAND: | **Flash** |
|---|---|
| SYNTAX: | `flash <cutout-ref> for <time> sec` |
| DESCRIPTION: | Flashes (i.e., rapidly changes from filled to unfilled) cut out `<cutout-ref>` over a specified period of time. Note that `<cutout-ref>`'s *fillcolor* attribute is used to determine the fill color. |
| EXAMPLE: | `flash mycutout for 1.5 sec` |

| COMMAND: | **Doconcurrent** |
|---|---|
| SYNTAX: | **doconcurrent**<br><animation-command-list><br>**enddoconcurrent;** |
| DESCRIPTION: | Executes multiple animation commands simultaneously. <animation-command-list> is a sequence of two or more animation commands. Note that doconcurrent statements may not be nested. |
| EXAMPLE: | doconcurrent<br>move my-cutout up 1.5 * height of my-cutout<br>move your-cutout down 1.5 * height of your-cutout<br>enddoconcurrent |

## Attributes of Storyboard Elements

On the following pages are terse summaries of all of the attributes defined on SALSA objects (cutouts, positions, and grids). The first table lists the attributes that all SALSA object types have in common. The subsequent three tables include attributes unique to each of the three SALSA object types. Note that all asterisked (*) attributes may be directly set by the programmer using the *set* command (*set <attribute> to <value>*).

### SALSA Object Attributes

The following table lists attributes that are common to all three SALSA object types.

| Attribute | Possible values | Default Value | Description |
|---|---|---|---|
| *name** | *text string* | *null* | A text identifier for the object |
| *id* | *text string* | — | System-defined unique identifier for the object |
| *data** | *text string* | "" (Empty string) | The data associated with the object, stored as a string. |
| *datavisible** | *true, false* | *true* | Whether or not to display the data associated with the object. |
| *dataposition** | *bottom-center, top-center*, center, left-center, right-center, ur-corner, lr-corner, ul-corner, ll-corner, custom | *below* | Where to display the object's data with respect to the object |
| *visible** | *true, false* | *true* | Whether or not the object is visible |
| *highlighted** | *true, false* | *false* | Whether or not the cutout is highlighted (using reverse-video). |
| *height* | — | — | Refers to the height of object; may only be changed through the *resize* animation command. |
| *width* | — | — | Refers to the width of object; may only be changed through the *resize* animation command. |
| *viewplane** | *integer* | *1* | The vertical view plane at which the object resides. The front-most storyboard element resides in plane 1; the further the view plane is from the front, the higher its number. |
| *bottom-center, top-center*, center, left-center, right-center, ur-corner, lr-corner, ul-corner, ll-corner, custom | — | — | Refer to various x,y locations in and around the object . Custom reference points may also be associated with an object |
| *refpoint** | *bottom-center, top-center, center, left-center, right-center, ur-corner, br-corner, ul-corner, bl-corner, custom* | *center* | The location that is actually referred to whenever the object is referred to (as in, e.g., *if mycutout is left-of yourcutout then. . .*). By default, this reference point is the geometric center of the cutout. It may be set to a custom refpoint (e.g., *elbow*) as well. |

## Cutout attributes

In addition to the SALSA object attributes listed above, the following attributes are defined on cutouts:

| Attribute | Possible values | Default Value | Description |
|---|---|---|---|
| *graphicrep** | *text string* | — | The name of the file containing the cutout's graphics; must be a .cut file |
| *outlinecolor** | *red, green, blue, purple, black, white, orange, gray, brown, yellow* | *black* | The outline color of the cutout |
| *fillcolor** | *same as above* | *white* | The fill color of the cutout |

## Position Attributes

In addition to the SALSA object attributes listed above, the following attributes are defined on positions:

| Attribute | Possible values | Default Value | Description |
|---|---|---|---|
| *xcoord* | *float* | *0.5* | The x-coordinate of the position |
| *ycoord* | *float* | *0.5* | The y-coordinate of the position |

## Grid Attributes

In addition to the SALSA object attributes listed above, the following attributes are defined on grids:

| Attribute | Possible values | Default Value | Description |
|---|---|---|---|
| *rows** | *integer* | *1* | The number of rows in the grid. |
| *columns** | *integer* | *5* | The number of columns in the grid. |
| *gridpoints** | *true, false* | *false* | Whether or not to show cell grid points. |
| *externalborder** | *true, false* | *false* | Whether or not to show the grid's outer border |
| *internalborder** | *true, false* | *false* | Whether or not to show the grid's outer border |
| *cellheight* | Expression involving heights and widths of cutouts | — | The height of a cell in the grid; all cells in a grid have the same height, which is determined by the tallest cutout contained by the grid. |
| *cellwidth* | Expression involving heights and widths of cutouts | — | The width of a cell in the grid; all cells in a grid have the same width, which is determined by the widest cutout contained by the grid. |

The following table lists a number of expressions that provide convenient access to individual grid positions and local positions around them.

| Expression | Description |
|---|---|
| `row of <cutout> of <grid>` | The numeric row index of a cutout that resides in a grid |
| `column of <cutout> of <grid>` | The numeric column index of a cutout that resides in a grid |
| `row of <gridpos>` | The numeric row index of a grid position |
| `column of <gridpos>` | The numeric column index of a grid position |
| `gridpos x,y of <grid>` | The position in the grid indexed by x and y (grid indices start at 1). |
| `cutout at <gridpos>` | the cutout presently located at a grid position—for example, *cutout at gridpos 1,1 of mygrid* |
| `<gridpos> is unoccupied` | Tests whether a particular grid position is *unoccupied*—that is, whether no cutout resides at that position. For example, *gridpos 1,1 of a is unoccupied* |
| `<gridpos> is occupied` | Tests whether a particular grid position is *occupied*—that is, whether a cutout resides at that position. For example, *gridpos 2,2 of a is occupied* |
| `gridpos in <grid> <spatial-kw> <position-expression>` | Refers to a grid position relative to a position in the storyboard—for example, *gridpos in mygrid left-of 0.3, 0.3* |
| `gridpos in <grid> <spatial-kw> <object-expression>` | Refers to a grid position relative to an object in the storyboard—for example, *gridpos in mygrid right-of cutout left-of foo* |
| `top-center of <gridpos>` | The top-center of the cell border of a grid position |
| `bottom-center of <gridpos>` | The bottom-center of the cell border of a grid position |
| `right-center of <gridpos>` | the right-center of the cell border of a grid position |
| `left-center of <gridpos>` | The left-center of the cell border of a grid position |
| `ul-corner of <gridpos>` | The upper left-hand corner of a grid position |
| `ur-corner of <gridpos>` | The upper right-hand corner of a grid position |
| `ll-corner of <gridpos>` | The lower left-hand corner of a grid position |
| `lr-corner of <gridpos>` | The lower right-hand corner of a grid position |

The following table lists a couple of expressions that provide a convenient means of referring to cutouts that presently reside in a grid:

| Expression | Description |
|---|---|
| `cutout in <grid> <spatial-kw> <position-expression>` | Refer to a cutout in a grid relative to a position expression, e.g., *cutout in a left-of left-center of foo* |
| `cutout in <grid> <spatial-kw> <object-expression>` | Refer to a cutout in a grid relative to an object expression, e.g., *cutout in a right-of cutout left-of foo* |

# APPENDIX G

## SAMPLE QUESTIONNAIRE FOR ASSESSING STUDENT ATTITUDES TOWARD AN ALGORITHMS COURSE

# CIS 315 Attitude Questionnaire

*Directions*. Each of the statements below reflects a certain feeling toward CIS 315. For each statement, please assess the extent of agreement between the feeling expressed by the statement, and your own personal feeling. Then circle the most appropriate level of agreement from the following five-step scale: *strongly disagree*, *disagree*, *undecided*, *agree*, and *strongly agree*.

| Statement | | | | | |
|---|---|---|---|---|---|
| 1. I feel as if I'm under a lot of stress in CIS 315. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 2. I do not like the theoretical side of computer science; it scares me to have to take CIS 315. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 3. Algorithmic problem solving, proofs of correctness, and efficiency analyses are interesting to me; I enjoy CIS 315. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 4. Algorithmic problem solving, correctness proofs, and efficiency analyses are fascinating and fun. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 5. I feel secure in doing algorithmic problem solving, proofs of correctness, and efficiency analyses; at the same time I find these activities stimulating. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 6. When trying to come up with an efficient algorithm to solve a given problem, or when trying to proof an algorithm's correctness or analyze its efficiency, I find that my mind goes blank and I am unable to think clearly. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |

| | | | | | |
|---|---|---|---|---|---|
| 7. | I feel insecure when attempting to design an algorithm, prove an algorithm's correctness, or analyze an algorithm's efficiency. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 8. | I become uncomfortable, restless, irritable, and impatient when working on CIS 315 problems. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 9. | I have a good feeling toward CIS 315. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 10. | When I'm learning about a new algorithm, or when I'm trying to follow a correctness proof or efficiency analysis, I feel that I get lost in the details. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 11. | I enjoy CIS 315 a great deal. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 12. | When I hear words like "proof of correctness" and "Big-O efficiency analysis," I have a feeling of dislike. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 13. | I approach CIS 315 with a feeling of hesitation, resulting from a fear of not being able to do algorithmic problem solving, proofs of correctness, and efficiency analyses. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 14. | I really like algorithmic problem solving, proofs of correctness, and efficiency analyses. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 15. | I have always enjoyed studying the theoretical and mathematical side of computer science. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |

| | | | | | |
|---|---|---|---|---|---|
| 16. | It makes me nervous to even think about having to come up with an efficient algorithm, correctness proof, or Big-O efficiency analysis in CIS 315. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 17. | I have never liked having to deal with the theoretical or mathematical side of computer science; CIS 315 is something I've been dreading. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 18. | I am happier in CIS 315 than in any other computer science class. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 19. | I feel at ease with the material covered in CIS 315, and I like it very much. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |
| 20. | I feel a definite positive reaction to what's being covered in CIS 315; it's enjoyable. | STRONGLY DISAGREE | DISAGREE | UNDECIDED | AGREE | STRONGLY AGREE |

# BIBLIOGRAPHY

Agre, P. E. (1997). Computation and human experience. New York: Cambridge University Press.

Aiken, L. R., & Dreger, R. M. (1961). The effect of attitudes on performance in mathematics. Journal of Educational Psychology 52, 19-24.

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

Badre, A., Beranek, M., Morris, J. M., & Stasko, J. T. (1991). Assessing program visualization systems as instructional aids (Tech. Rep. No. GIT-GVU-91-23). Atlanta: Graphics, Visualization, and Usability Center, Georgia Institute of Technology.

Baecker, R. (1975). Two systems which produce animated representations of the execution of computer programs. SIGCSE Bulletin 7(1), 158-167.

Baecker, R. (1998). Sorting out sorting: A case study of software visualization for teaching computer science. In M. Brown, J. Domingue, B. Price, & J. Stasko (Eds.), Software visualization: Programming as a multimedia experience (pp. 369-382). Cambridge, MA: The MIT Press.

Baecker, R., & Sherman, D. (1981). Sorting Out Sorting [16 mm color sound film]. (Excerpted and "reprinted" in SIGGRAPH Video Review 7, 1983. Available from Morgan Kaufmann Publishers, Los Altos, CA)

Bazik, J., Tamassia, R., Reiss, S., & van Dam, A. (1998). Software visualization in teaching at Brown University. In M. Brown, J. Domingue, B. Price, & J. Stasko (Eds.), Software visualization: Programming as a multimedia experience (pp. 383-398). Cambridge, MA: The MIT Press.

Bell, B., Rieman, J., & Lewis, C. (1991). Usability testing of a graphical programming system: Things we missed in a programming walkthrough. In Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems (pp. 7-12). New York: ACM Press.

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: Erlbaum.

Boster, J. S. (1985). "Requiem for the omniscient informant": There's life in the old girl yet. In J. W. D. Dougherty (Ed.), Directions in cognitive anthropology (pp. 177-197). Urbana: University of Illinois Press.

Bower, G. H., & Clark, M. C. (1969). Narrative stories as mediators for serial learning. Psychonomic Science 14, 181-182.

Brown, D. R., & Vander Zanden, B. (1998). The Whiteboard environment: An electronic sketchpad for data structure design and algorithm description. In Proceedings of the 1998 IEEE Symposium on Visual Languages (pp. 288-295). Los Alamitos, CA: IEEE Computer Society Press.

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

Brown, M. H., & Hershberger, J. (1992). Color and sound in algorithm animation. IEEE Computer 25(12), 52-63.

Brown, M. H., & Najork, M. A. (1993). Algorithm animation using 3D interactive graphics. (Tech. Rep. No. 110a). Palo Alto, CA: DEC Systems Research Center.

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

Brown, M. H., & Sedgewick, R. (1985). Techniques for algorithm animation. IEEE Software 2(1), 28-39.

Byrne, M. D., Catrambone, R., & Stasko, J. T. (1996). Do algorithm animations aid learning? (Tech. Rep. No. GIT-GVU-96-18). Atlanta: Graphics, Visualization, and Usability Center, Georgia Institute of Technology.

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

Citrin, W., & Gurka, J. (1996). A low-overhead technique for dynamic blackboarding using morphing technology. Computers & Education 26, 189-196.

Cobb, P. (1996). Where is the mind: A coordination of sociocultural and cognitive constructivist perspectives. In C. T. Fosnot (Ed.), Constructivism: Theory, perspectives, and practice (pp. 34-52). New York: Teachers College Press.

Cohen, J. (1988). Statistical power analysis for the behavorial sciences. Hillsdale, NJ: Lawrence Erlbaum.

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

Cox, K. C., & Roman, G. C. (1994). An evaluation of the Pavane visualization system. Technical report No. WUCS-94-09. Department of Computer Science, Washington University in St. Louis, St. Louis, MO.

Denning, P. J. (1989). Computing as a discipline. Communications of the ACM 32(1), 9-23.

Doerry, E. (1995). An empirical comparison of copresent and technologically-mediated interaction based on communicative breakdown. Unpublished doctoral dissertation, University of Oregon, Eugene.

Dominigue, J., Price, B. A., & Eisenstadt, M. (1992). A framework for describing and implementing software visualization systems. In Proceedings of Graphics Interface '92 (pp. 53-60). Palo Alto, CA: Morgan Kaufmann.

Douglas, S. A., Doerry, E., & Novick, D. (1992). QUICK: A tool for graphical user interface construction. The Visual Computer 8(2), 117-133.

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.

Douglas, S. A., Novick, D. G., & Tomlin, R. S. (1987, June). Consistency and variation in spatial reference. Paper presented at the Ninth International Conference on Cognitive Science, Seattle, WA.

Duisberg, R. (1987a). Visual programming of program visualizations. In Proceedings of the IEEE 1987 Visual Language Workshop . Los Alamitos, CA: IEEE Computer Society Press.

Duisberg, R. A. (1987b). Animation using temporal constraints: An overview of the animus system. Human-Computer Interaction 3(3), 275-307.

Eisenstadt, M., Price, B. A., & Dominique, J. (1993). Software visualization as a pedagogical tool. Instructional Science 21, 335-364.

Erwig, M. (1991). DEAL - A language for depicting algorithms. In Proceedings of the 1991 Workshop on Visual Languages (pp. 184-185). Los Alamitos, CA: IEEE Computer Society Press.

Estroff, S. (1981). Making it crazy: An ethnography of psychiatric clients in an american community. Berkeley: University of California Press.

Gloor, P. A. (1992). AACE - algorithm animation for computer science education. In Proceedings of the 1992 IEEE Workshop on Visual Languages (pp. 25-31). Los Alamitos, CA: IEEE Computer Society Press.

Griss, S. (1999, spring). Reading, writing, and jumping around. Smith Alumnae Quarterly, pp. 24-31.

Gruber, H. E., & Voneche, J. J. (Eds.). (1977). The essential Piaget: An interpretive reference and guide. New York: Basic Books

Gurka, J. S. (1996). Pedagogic aspects of algorithm animation. Unpublished doctoral dissertation, University of Colorado, Boulder.

Gurka, J. S., & Citrin, W. (1996). Testing effectiveness of algorithm animation. In Proceedings of the 1996 IEEE Symposium on Visual Languages (pp. 182-189). Los Alamitos, CA: IEEE Computer Society Press.

Hein, G. E. (1991). The museum and the needs of the people. Paper presented at the International Committee of Museum Educators Conference, Jerusalem, Israel.

Helttula, E., Hyrskykari, A., & Raiha, K.-J. (1989). Graphical specification of algorithm animations with ALLADDIN. Procedings of the 22nd Annual Conference on Systems Sciences, 892-901.

Hill, R. D., Allen, C., & McWhorter, P. (1991). Stories as a mnemonic aid for older learners. Psychology and Aging 6(3), 484-486.

Hundhausen, C. D. (1997). A meta-study of software visualization effectiveness. Unpublished manuscript, University of Oregon, Department of Computer and Information Science.

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.

Kehoe, C. M., & Stasko, J. T. (1996). Using animations to learn about algorithms: An ethnographic case study. (Tech. Rep. No. GIT-GVU-96-20). Atlanta: Graphics, Visualization, and Usability Center, Georgia Institute of Technology..

Knox, D. et al. (1996). Use of laboratories in computer science: Guidelines for good practice (Report of the Working Group on Computing Laboratories). SIGCSE Bulletin 28, 167-181.

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

Landay, J.A., & Myers, B.A. (1995). Interactive sketching for the early Stages of user interface design. In Proceedings of ACM CHI '95 Conference on Human Factors in Computing Systems (pp. 43-50). New York: ACM Press.

Lave, J. (1997). The culture of acquisition and the practice of understanding. In D. Kirshner & J. Whitson (Eds.), Situated cognition: Social, semiotic, and psychological perspectives (pp. 17-35). Mahwah, NJ: Erlbaum.

Lave, J. (1993). The practice of learning. In S. Chaiklin & J. Lave (Eds.), Understanding Practice: Perspectives on activity and context (pp. 3-32). Cambridge: Cambridge University Press.

Lave, J. (1988). Cognition in practice. Cambridge, MA: Cambridge University Press.

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 doctoral dissertation, Georgia Institute of Technology, Atlanta.

Lawrence, A. W., Badre, A. N., & Stasko, J. T. (1994). Empirically evaluating the use of animations to teach algorithms. In Proceedings of the 1994 IEEE Symposium on Visual Languages (pp. 48-54). Los Alamitos, CA: IEEE Computer Society Press.

Lieberman, H. (1989). A three-dimensional representation for program execution. In Proceedings of the 1989 Workshop on Visual Languages (pp. 111-116). Los Alamitos, CA: IEEE Computer Society Press.

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.

Martin, D. W. (1996). Doing psychology experiments. San Francisco: Brooks/Cole.

Mayer, R. E., & Anderson, R. B. (1991). Animations need narrations: An experimental test of a dual-coding hypothesis. Journal of Educational Psychology 83(4), 484-490.

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

Michail, A. (1996). Teaching binary tree algorithms through visual programming. In Proceedings of the 12th IEEE Symposium on Visual Languages (pp. 38-45). Los Alamitos, CA: IEEE Computer Society Press.

Mukherjea, S., & Stasko, J. T. (1994). Toward visual debugging: Integrating algorithm animation capabilities within a source-level debugger. ACM Transactions on Computer-Human Interaction 1(3), 215-244.

Myers, B. A. (1983). Incense: A system for displaying data structures. Computer Graphics 17(3), 115-125.

Najork, M. A., & Brown, M. H. (1995). Obliq-3D: A high-level, fast-turnaround 3D animation system. IEEE Transactions on Visualization and Computer Graphics 1(2), 175-193.

Naps, T.L. (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.

Naps, T. L., & Hundhausen, C. D. (1991). The evolution of an algorithm visualization system. Proceedings of the 24th Annual Small College Computing Symposium, 259-266.

Newell, A. (1980). Physical symbol systems. Cognitive Science 4, 135-183.

Norman, D. A., & Draper, S. W. (Eds.). (1986). User-centered system design . Hillsdale, NJ: Lawrence Erlbaum Associates.

Paivio, A. (1969). Mental imagery in associative learning and memory. Psychological Review 76, 241-260.

Paivio, A. (1983). The empirical case for dual coding. In J. C. Yuille (Ed.), Imagery, memory, and cognition: Essays in honor of Allan Paivio . Hillsdale, NJ: Lawrence Erlbaum Associates.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. New York: Basic Books.

Price, B. A., Baecker, R. M., & Small, I. S. (1993). A principled taxonomy of software visualization. Journal of Visual Languages and Computing 4(3), 211–266.

Reddy, M. J. (1977). The conduit metaphor--A case of frame conflict in our language about language. In A. Ortony (Ed.), Metaphor and thought (pp. 284-324). New York: Cambridge University Press.

Resnick, L. B. (1989). Introduction. In L. B. Resnick (Ed.), Knowing, learning, and instruction: Essays in honor of Robert Glaser (pp. 1-24). Hillsdale, NJ: Erlbaum.

Rogers, J. L., Howard, K. I., & Vessey, J. T. (1993). Using significance tests to evaluate equivalence between two experimental groups. Psychological Bulletin 113(3), 553-565.

Roman, G. C., & Cox, K. C. (1993). A taxonomy of program visualization systems. IEEE Computer 26(12), 11-24.

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.

Romney, A. K., Weller, S. C., & Batchelder, W. H. (1986). Culture as consensus: A theory of culture and informant accuracy. American Anthropologist 88(2), 313-338.

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

Schumann, J., Strothotte, T., Raab, A., & Laser, S. (1996). Assessing the effect of non-photorealistic rendered images in CAD. In Human Factors in Computing Systmes: CHI 96 Conference Proceedings (pp. 35-41). New York: ACM Press.

Sedgewick, R. (1988). Algorithms. Reading, MA: Addison-Wesley.

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

Sigle, J. (1990). Dynamic display of computing processes. SIGCSE Bulletin 22(2), 2-4.

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 doctoral dissertation, University of California-Davis.

Stasko, J.T. (1989). TANGO: A framework and system for algorithm animation. Unpublished doctoral dissertation, Brown University, Providence, RI.

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

Stasko, J. T. (1991). Using direct manipulation to build algorithm animations by demonstration. In Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems (pp. 307-314). New York: ACM Press.

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.

Stasko, J.T., 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.T., & 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., & Wehrli, J. F. (1993). Three-dimensional computation visualization. In IEEE Symposium on Visual Languages (pp. 100-107). Los Alamitos, CA: IEEE Computer Society Press.

Suchman, L. A. (1987). Plans and situated actions: The problem of human-computer communication. New York: Cambridge University Press.

Turing, A. M. (1950). Computing machinery and intelligence. Mind 59, 433–460.

van de Kant, M., Wilson, S., Bekker, M., Johnson, H., & Johnson, P. (1998). PatchWork: A software tool for early design. In Human Factors in Computing Systems: CHI 98 Summary (pp. 221-222). New York: ACM Press.

Wenger, E. (1987). Artificial intelligence and tutoring systems. Los Altos, CA: Morgan Kaufmann.

Wenger, E. (1998). Communities of practice : Learning, meaning, and identity. Cambridge: Cambridge University Press.

Wilson, J., Katz, I. R., Ingargiola, G., Aiken, R., & Hoskin, N. (1995). Students' use of animations for algorithm understanding. In Proceedings of the CHI '95 Conference on Human Factors in Computing Systems (pp. 238-239). New York: ACM Press.

Wolcott, H. F. (1992). Posturing in qualitative inquiry. In M. D. LeCompte, W. L. Millroy, & J. Preissle (Eds.), The Handbook of Qualitative Research in Education (pp. 3-52). San Diego: Academic Press.