Shifting From “High Fidelity” to “Low Fidelity” Algorithm Visualization Technology

Christopher Hundhausen
Laboratory for Interactive Learning Technologies
Information and Computer Sciences Department
University of Hawai`i at Manoa
Honolulu, HI 96822
hundhaus@hawaii.edu

Sarah Douglas
Human-Computer Interaction Lab
Computer and Information Science Department
University of Oregon
Eugene, OR 97403–1202
douglas@cs.uoregon.edu

ABSTRACT
Traditional algorithm visualization software supports the creation of “high fidelity” visualizations, which depict the target algorithm for arbitrary input, and have the polished look of textbook figures. Drawing on the findings of ethnographic studies we conducted in an undergraduate algorithms course, we have developed SALSA/ALVIS, a markedly different kind of algorithm visualization software that enables students to construct and present their own “low fidelity” visualizations. Unlike “high fidelity” visualizations, “low fidelity” visualizations depict the target algorithm for a few, carefully-selected input data sets, and have an unpolished look. In addition, they can be viewed both forwards and *backwards*, and dynamically marked up and modified, making them well-suited for mediating student-instructor discussions about algorithms.

Keywords
Algorithm visualization, computer science education

INTRODUCTION
Algorithm visualization software graphically depicts how computer algorithms work. Traditionally, computer science instructors have used the software to construct visualizations that are later used either as visual aids in lectures, or as the basis for interactive labs (e.g., [1]). More recently, some computer science educators have advocated using algorithm visualization software as the basis for “visualization assignments,” in which students construct their own visualizations of the algorithms under study [5].

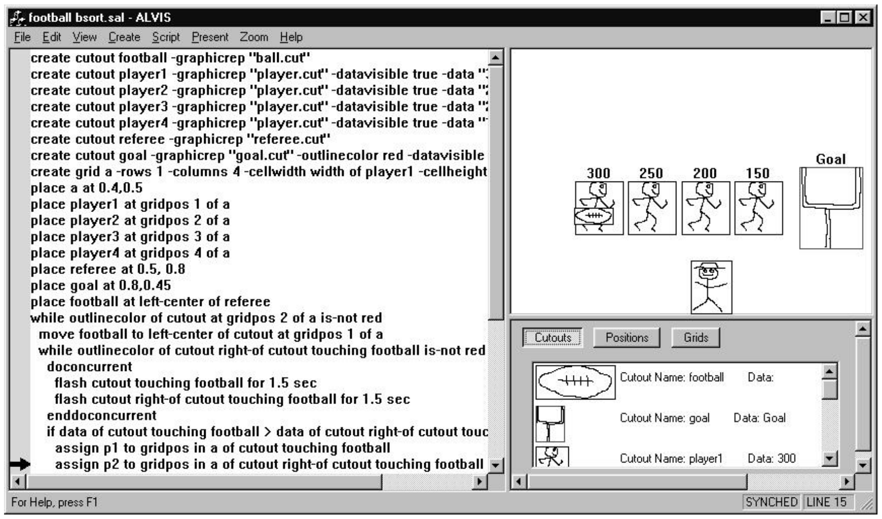
Inspired by social learning theory [4], we have explored a teaching approach that takes such “visualization assignments” one step further by having students present their own visualizations to their instructor and peers for feedback and discussion [3]. To better understand this approach, we conducted a series of ethnographic studies in a junior-level algorithms course that included such assignments. Our findings have led us to distinguish between “high fidelity” and “low fidelity” algorithm visualization technology. In this paper, we develop that distinction, and we introduce a prototype “low fidelity” algorithm visualization system rooted in our empirical findings.

STUDY I: “HIGH FIDELITY” VISUALIZATIONS
In the first of our ethnographic studies, students used Stasko’s *Samba* animation package [5] to construct “high fidelity” visualizations. By “high fidelity,” we mean visualizations that depict the target algorithm for general input, and that, because they are generated automatically as a by-product of algorithm execution, tend to have the polished look of textbook figures.

In this first study, three key findings are noteworthy. First, students spent over 33 hours on average constructing a single high fidelity visualization. They spent most of that time steeped in *low-level graphics programming*, which is largely irrelevant within an algorithms course. Second, in students’ subsequent presentations, high fidelity visualizations tended to stimulate discussions about *implementation details*—e.g., how a particular aspect of a visualization was implemented. Third, in response to audience feedback, students often wanted to back up and re-present parts of their visualizations, or to dynamically mark-up and modify them. However, high fidelity software like Samba cannot support interactive presentations in this way.

STUDY II: “LOW FIDELITY” VISUALIZATIONS
In the second of our ethnographic studies, students were required to use simple art supplies (e.g., pens, paper) to construct and present “low fidelity” visualizations (as in [2]). By “low fidelity,” we mean visualizations that work for a few, carefully selected input data sets, and, because they are hand-constructed, have a rough, unpolished look.

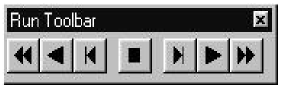
In this second study, three key findings stand out. First, students spent only about six hours on average constructing a single, low fidelity visualization. For most of that time, students focused on the target algorithm’s procedural behavior, and how they might best communicate it. Second, rather than stimulating discussions about implementation details, low fidelity visualizations tended to mediate discussions about the *underlying algorithm*, and about how the visualizations might bring out its behavior more clearly. Third, students could readily mark-up and dynamically modify their visualizations in response to audience feedback. As a result, presentations tended to be more interactive and collaborative.



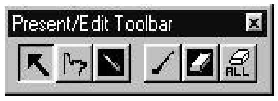
```
create cutout football -graphicrep "ball.cut"
create cutout player1 -graphicrep "player.cut" -datavisible true -data "
create cutout player2 -graphicrep "player.cut" -datavisible true -data "
create cutout player3 -graphicrep "player.cut" -datavisible true -data "
create cutout player4 -graphicrep "player.cut" -datavisible true -data "
create cutout referee -graphicrep "referee.cut"
create cutout goal -graphicrep "goal.cut" -outlinecolor red -datavisible
create grid a -rows 1 -columns 4 -cellwidth width of player1 -cellheight
place a at 0.4,0.5
place player1 at gridpos 1 of a
place player2 at gridpos 2 of a
place player3 at gridpos 3 of a
place player4 at gridpos 4 of a
place referee at 0.5, 0.8
place goal at 0.8,0.45
place football at left-center of referee
while outlinecolor of cutout at gridpos 2 of a is-not red
 move football to left-center of cutout at gridpos 1 of a
 while outlinecolor of cutout right-of cutout touching football is-not red
  doconcurrent
   flash cutout touching football for 1.5 sec
   flash cutout right-of cutout touching football for 1.5 sec
  enddoconcurrent
  if data of cutout touching football > data of cutout right-of cutout touc
   assign p1 to gridpos in a of cutout touching football
   assign p2 to gridpos in a of cutout right-of cutout touching football
```

	Cutout Name: football	Data:
	Cutout Name: goal	Data: Goal
	Cutout Name: player1	Data: 300

(a) Snapshot of a Session with ALVIS



(b) The Execution Control Interface



(c) The Presentation Interface

Figure 1: The ALVIS Low Fidelity Visualization Environment

A "LOW FIDELITY" LANGUAGE AND SYSTEM

To illustrate the design implications of our empirical findings, we have developed SALSA (Spatial Algorithmic Language for StoryboArding), a high-level, interpreted language for specifying low-fidelity visualizations, along with ALVIS (ALgorithm VIsualization Storyboarder), a direct-manipulation environment for programming in SALSA ([3], ch. 7).

Figure 1a presents a snapshot of a sample session with the ALVIS environment. In the *script* region (left), the user is editing a SALSA script that generates the "football" Bubblesort visualization (see [2]). Unlike high fidelity visualization software, which requires users to create objects in terms of Cartesian coordinates, ALVIS enables users to create objects simply by sketching them in a graphics editor (not shown), and dragging-and-dropping them from the *created objects* region (bottom-right) to the *animation* region (top-right). For example, ALVIS generated the statement *place referee at 0.5, 0.8* in response to the user's dragging-and-dropping the referee to its current position.

In prior empirical studies [2], we observed that authors of low fidelity visualizations tend to execute their visualizations by maintaining important *spatial relations* among visualization objects. The hallmark of SALSA is its support for referencing visualization objects and programming visualization logic in terms of such spatial relations. For example, the script depicted in Figure 1a flashes the two players currently being compared. Those two players are referenced solely in terms of their spatial relationships to the football (*flash cutout touching football* and *flash cutout right-of cutout touching football*).

Finally, ALVIS's presentation interface supports three key features that distinguish it from high-fidelity visualization systems. First, ALVIS's execution interface (Figure 1b) supports both forwards and *backwards* execution, enabling the user to unwind and re-present portions of the animation at the request of the audience. Second, ALVIS's presentation interface (Figure 1c) provides several tools that support interactive presentation, including a pen and eraser for dynamically marking up a visualization as it is being presented. Third, in response to audience feedback, the user may dynamically modify the visualization simply by inserting or removing SALSA code at the point at which the script is presently halted (denoted by the black arrow to the left of the script in Figure 1a).

STATUS AND FUTURE WORK

SALSA and ALVIS are working research prototypes. In future work, we plan to subject them to iterative usability testing in order to evaluate and improve on their design. Our goal is eventually to release them into the public domain. Consult http://lilt.ics.hawaii.edu/~hundhaus/dis for up-to-date information on this project.

ACKNOWLEDGEMENTS

Thanks to Hank Bennett and Ted Kirkpatrick for invaluable assistance in programming and debugging ALVIS.

REFERENCES

1. Brown, M.H. *Algorithm animation*. Cambridge: The MIT Press, 1988.
2. Douglas, S.A., Hundhausen, C.D., & McKeown, D. Exploring human visualization of computer algorithms. In *Proc. 1996 Graphics Interface Conference* (pp. 9–16). Toronto: Canadian Graphics Society, 1996.
3. Hundhausen, C.D. Toward Effective Algorithm Visualization Artifacts: Designing for Participation and Communication in an Undergraduate Algorithms Course. Unpublished Doctoral Dissertation (Tech. Rep. No. CIS-TR-99-07). Eugene: Dept. of Computer and Info. Science, U. of Oregon, 1999.
4. Lave, J., & Wenger, E. *Situated learning: Legitimate peripheral participation*. New York: Cambridge U. Press, 1991.
5. Stasko, J. T. Using student-built animations as learning aids. In *Proc. ACM Technical Symposium on Computer Science Education* (pp. 25-29). New York: ACM Press, 1997.