Evaluation of Effectiveness of Incorporation of Computer Vision into Undergraduate Data Structures

Sudeep Sarkar (sarkar@csee.usf.edu)

***Abstract*—In the past [17][18], we have proposed a scheme for incorporating computer vision tasks and algorithms into undergraduate data structures courses to enrich the experience and enhance the learning process. We have found that introductory computer vision can be easily integrated without detracting from the original goal of teaching data structures. Some of the image-related tasks, such as connected component labeling and binary image manipulation, offer a natural way to introduce basic data structures such as arrays, queues, stacks, trees, and hash tables. The particular advantage of this integrated strategy is that it exposes the students to image related manipulations at an early stage of the curriculum, furthermore, it facilitates the understanding of data structure concepts by providing a visual context. We have experimented with such an integration strategy using a set of programming assignments. These assignments can be incorporated in existing data structures courses with low time and software overheads. In this paper, we present quantitative analysis of the effectiveness of these assignments using pre- and post-assignment tests. We also consider student performance and their evaluations of the assignments. Our analysis suggests that the assignments significantly improved the understanding of standard data structures and basic software design principles.**

***Index Terms*—Computer vision, image computations, data structures, undergraduate computer science education.**

I. Introduction

THE past couple of decades have witnessed a tremendous increase in imaging modalities that range from hand-held home video cameras to systems that can image the functioning brain to extraterrestrial probes that beam back to earth tremendous amounts of image data. This has created a demand for software that can process, improve, manipulate, search, and analyze these images. It is easy to foresee a future when this demand would not be restricted just to the scientific or medical community but would extend to the general home consumer. The easy accessibility of (digital) cameras and the Internet will only fuel this demand. Responding to this demand, it is imperative that today's computer science graduate is familiar with not only the traditional core computer science, but also with manipulating and using image data. Images are unlike regular data. For one, each unit of image data is usually very large. Another aspect is the fact that while it is easy for a human to interpret image data, automated computer-based understanding of images is difficult. This can raise impractical expectations about what information can be realistically extracted from images. Hence, it is necessary that future graduates are familiar with the nature of images as information sources in many different contexts. The traditional strategy of introducing these topics in image processing [3] or computer vision [10][15] as special elective courses is no longer adequate.

Apart from the future demand for image related expertise, images offer an excellent visual medium [14] for illustrating computer science concepts. For example, the nature of breadth-first and depth-first searches can be demonstrated visually by the way the image pixels are marked during connected component labeling. In our preliminary experience with integrating image algorithms in data structures courses, we have found that students like the visual nature of the assignments. Another excellent use of image processing techniques is in introducing computational complexity issues. For instance, time and space complexities of stereo image processing strategies for extracting depth from images vary from low-order polynomial to exponential ones, depending the constraints and assumptions that one uses. The impact of decrease or increase of computational complexity can be effectively demonstrated just by increasing the size of the images, which is easy to do.

Previous suggestions for improving undergraduate data structure and algorithms education have mainly been concerned with visualization aspects. Some excellent suggestions appear in [12][19][22]. Augmentation of a typical data structures course by a laboratory course has also been found to be effective [11]. There have been demonstrated advantages in teaching the topic of sorting in data structures course using hands-on approaches [5]. It has also been shown that multiple visual representations can enhance the understanding of data structures [8]. In the past [17][18], we have advocated enhancing *existing* undergraduate data structures courses by introducing elements of image computations into such courses. Such an integration of image processing has also been suggested by the IPT project at the University of Arizona but for enhancing the science and mathematics curricula of community colleges and secondary schools [6][7].

In this paper, we document our effort in evaluating the effectiveness of our effort in integrating image computations into data structures. There are various possible evaluation strategies [1], such as dividing the class into two sections, control and experimental, as was done in [4][5] or adopting an extensive longitudinal strategy that involve follow-ups over a period of years [2]. We adopted a strategy that relied on pre-,

The National Science Foundation Grant DUE 9980832 supported this work.

Sudeep Sarkar is with the Computer Science and Engineering Department; 4202 E Fowler Ave., ENB 118; University of South Florida; Tampa 33620; (phone: (813) 974 2113; fax: (813) 974-5456; email: sarkar@csee.usf.edu).

post-tests, and student perceptions of the effectiveness of the assignments. The hypothesis of our evaluation was

> Image-based programming assignments improve the understanding of standard data structure concepts.

In the next section, we describe the different elements of the methodology, which is followed by the analysis section. We conclude with a discussion of the other possible evaluative questions that we would like to answer in future.

II. Methods

A. Structure of the Course

The data structures course at USF is a lower-level introductory course, which the students typically take in their junior years. The pre-requisites for this course include the Introduction to Program Design and the Introduction to Computer Architecture courses, which are the two gate courses in the department. The students are expected to be comfortable with programming constructs such as iterations loops, string processing, and file I/O. They are also sometimes exposed to recursions and linked lists in the pre-requisite courses. The programming language of choice for all the introductory courses, including data structures, is C.

We evaluated the image-based materials in the data structures course during the Spring 2001 semester. The official (supported) OS for the programming assignments was Unix (Sun-Solaris) and the language was C. However, a number of students first completed their assignments on a PC and then migrated them to Unix without much effort. The Unix-based image display software, ImageMagick, was available for viewing images. Students, on their own, were also able to view images on PCs by using many of the image editors/viewers that are available, e.g. MS-Paint, Explorer, and Netscape. The textbook used for the course was "Data Structures and Program Design in C," by Kruse, Tondo, and Leung (Prentice Hall).

Only one lecture period was reserved for introducing the student to image related background that discussed concepts of image pixels, image sampling, gray level quantization, the Portable Gray Map format, how to convert to other image formats, and how to display images on screen. All other lectures were on traditional data structure topics. We also distributed a handout discussing the image-related background materials so that the students can readily refer to it throughout the semester.

B. Data and Collection Methods

There were three programming assignments, three examinations, and ten quizzes, spread throughout the semester. For the materials of each of the three assignments, we had in-class tests before and after the assignments. Some of these were short quizzes and some were part of the three main examinations. The student performances in these tests constitute first part of the data that were analyzed. The topics of the three pre- and post-tests were as follows:

1. The concepts of information hiding and modularity associated with the design of Abstract Data Types (ADTs).
2. The stack, queue, & list of lists ADTs and their use in depth- and breadth-first searches.
3. The graph ADT and the task of connected component labeling of a graph.

Each set of pre- and post-tests were, of course, not identical, however, they tested the same materials, which were the focus of the respective assignments, as listed above.

The second source of evaluative data is the student performances in the programming assignments themselves. We evaluated each programming assignment in terms of:

1. *Documentation*, which was expected to include flowchart, description of use, results on images, and list of programming bugs encountered. It was evaluated out of 5 points.
2. *Source code*, which was expected to be properly indented, commented, have meaningful function and variable names, and be modular. The evaluation was out of 5 points.
3. *Execution of the code*, which was evaluated out of 10 points.

A detailed grading rubric was used to enhance consistency of grading across assignments and students. The students were provided with a copy of the grading rubric towards the beginning of the semester.

The third source of evaluative data is the student evaluations of the assignments. At the end of the semester, we asked the students to anonymously evaluate the use of the image-related materials in the course in terms of the following aspects

1. Number of hours spent on each assignment.
2. The sufficiency of the number of assignments.
3. The sufficiency of the image-related support materials.
4. Whether they would recommend the course to others.
5. How much the assignments facilitated their understanding of the data structures studied in class.
6. How much the assignments improved their programming skills.
7. How much the assignments helped in the understanding and learning of: information hiding, ADT use and implementation rule, makefiles, code modularity, new ADT design, stacks & queues, list of lists, graphs, depth- and breadth-first searches, basic image processing algorithms, documentation skills,

and importance of code commenting.

The students also had an option to add additional written comments regarding the inclusion of the image-related tasks in the course.

C. Assignments

The general context of the assignments was the task of segmenting walking persons in motion sequences. The goal was to detect image pixels that have moved. The adopted technique relied on frame differencing, following by morphological processing, connected component labeling to detect motion blobs, and proximity-based grouping of the motion blobs. Figure 1 shows the flow of information between the various data structure and algorithms that were used to implement the motion detection task. The white blocks correspond to the first assignment. The white blocks with thick borders are part of the second assignment. The shaded blocks with thick borders are the topic of the third assignment. Notice how each of the assignments uses the products from the others to build one large piece of software, accomplishing a non-trivial task.

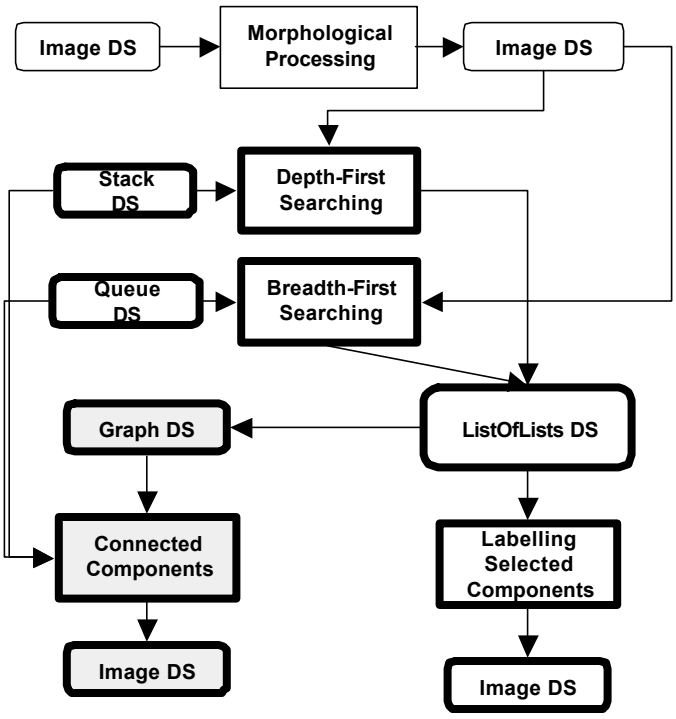

Figure 1: Flow of information between data structures and algorithms that are part of the assignments. The white, white with thick border, and shaded with thick border blocks correspond to the first, second, and third assignments, respectively.

Assignment 1: The objectives were:

1. to familiarize the students with the design of a simple array based data,
2. to impress upon the students the principles of Abstract Data Type (ADT) use and design,
3. to emphasize the principles of information hiding,
4. to introduce simple image processing algorithms such as image differencing, thresholding, and the morphological operations of erosion, dilation, opening and closing.

The image related task for this assignment was to detect motion in images by simple image differencing, followed by cleanup using morphological operations on the thresholded difference image.

As part of their programming, the students had to implement the image abstract data type (ADT) using 1D arrays as per given specifications. The ADT had to support functions to read images, save images, set and get image size, set and get pixel values, indicate whether a location is inside or outside the image boundary, and free or delete the ADT. They also had to write an application program that interacted with the Image ADT to accomplish the tasks of (i) subtracting two images, (ii) thresholding two images, and (iii) performing the morphological opening and closing operations.

Assignment 2: The objectives of this assignment were

1. to use the stack and the queue ADTs,
2. to design a new ADT, namely the *ListOfLists* ADT,
3. to enhance the understanding of depth-first and breadth-first searches,
4. to teach connected component based labeling of binary images,
5. to introduce source code management in Unix using Makefiles, and also
6. to emphasize code reuse by insisting that this assignment use the code from the first one.

The image related task for this assignment was to extract connected motion pixels in the image, thus extracting motion blobs.

The input to this assignment was the morphologically processed difference image from the first assignment. The output was an image with each connected black region labeled by a unique number. As part of their programming, the students used the stack and the queue implementations that were provided in the book and specified, designed, and implemented the new ListOfList ADT. They then used these ADTs to implement the depth-first and breadth-first based connected component labeling of the input (from the first assignment) image. Each individual component of the source code was divided into different files, which were managed using the Unix Makefile facility.

Assignment 3: The objectives of this assignment were:

1. to specify, design, and implement the Graph ADT *using*

the List ADT,
2. to learn about finding connected components of graphs, and, of course,
3. to reinforce good programming habits by insisting that this assignment be built upon the first two assignments.

The task was to form groups of motion blobs, which were found in the second assignment, most likely to belong one person. The adopted strategy was proximity-based grouping: blobs that are close together most likely belong to the same person. This proximity based grouping strategy was implemented using the Graph ADT. The nodes of the Graph ADT represented the individual motion blobs, and two nodes whose corresponding regions were less than a user specified distance were connected by a link. Connected components of the graph, identified by depth-first or breadth-first search, form the groups of regions, which constitute the output.

The input to this assignment was the ListOfLists data structure, built as part of the second assignment based on the connected pixels in the morphologically processed difference image. As part of the programming, the students designed and implemented the GraphADT using the List ADT to represent the adjacency lists. They also wrote an application program to construct the graph representation from the ListOfLists representation and to perform connected component extraction from the constructed graph.

III. Analysis

A. Analysis of Pre- and Post-Test Scores

There were three sets of pre- and post-test scores, corresponding to the three assignments, for each of the 24 students in the class. We first consider if there were significant differences between the pre- and post-test scores.

Table 1: Mean scores on pre- and post-tests for the three assignments

Assignments	Mean Pre-Test Score	Mean Post-Test Score
One	53.5	60.7
Two	55.6	66.1
Three	62.3	72.6

Table 1 lists the mean pre- and post-test scores for each of the assignments. There is an obvious increase in the mean scores. The question is: are the differences statistically significant? There are three possible sources of variations: (i) the time (pre and post) when the tests were administered with respect to the assignments, which we denote by TIME, (ii) the individual students, denoted here by STUDENTS, and (iii) the assignments themselves, which we denote by TASK. Table 2 lists the Analysis of Variance (ANOVA) results. ANOVA is a statistical tool that allows us to determine the extent to which different sources contribute to the overall variation in the scores. The first column of the table lists the sources of variation, whose corresponding degrees of freedom (DF) are listed in the second column. The third and the fourth columns list the sum of squares (SS) and the mean squared (MeanSq) values. The fifth column shows the corresponding F-values, which are the ratios of the variation due to the individual sources to that of random variations in the data. Higher the F-value, higher is the contribution of that factor to the overall variation. The last column lists the significance of F-values, in terms of the probability of rejecting the NULL hypothesis that the corresponding source does not contribute to the overall variation. With a 0.05 significance threshold, we can say with 95% confidence that both the TIME and the STUDENT sources contribute significantly to the overall variation. In other words, the differences in the pre- and the post-test scores are significantly different. The conclusion that the scores of the students differed significantly is, of course, not surprising. However, it is interesting to note that the scores did not vary significantly over the three TASKS, which suggests that the three assignments contributed equally to the variations in the scores.

Table 2: Analysis of Variance (ANOVA) of the pre- and post-assignment tests scores. The independent factors are: TIME when the tests were administered, the STUDENTS, and the TASK tested upon.

Source	DF	SS	MeanSq	F-value	Pr > F
TIME	1	2678.1	2678.1	4.33	**0.0397**
STUDENT	23	26402.1	1147.9	1.85	**0.0175**
TASK	2	2401.0	1200.5	1.94	0.1485

Table 3 shows the mean and median values of the differences between the post- and pre-test scores for each of the three assignments. As comparison, we list the difference in scores of a topic, namely binary trees, that was tested twice over the semester, but was not part of any assignment. This is our control. Note that all the differences are positive, which denotes improvement in scores over time. The mean improvements for the assignment topics are almost double to that for the control. The median scores, which are robust measures, are in the range from 8% to 17% and are markedly better than that for the control, which has a median score value of 0%.

Table 3: Mean and median of the difference in pre- and post-assignment performances

Statistic	Assign 1	Assign 2	Assign 3	Control
Mean	7.2%	9.5%	10.3%	4.0%
Median	17.0%	8.0%	10.0%	0.0%

B. Analysis of Assignment Grades

In this section we look at how the students performed in the assignments themselves. The last column of Table 4 lists the mean scores for each of the three assignments. The total score is a number of three component scores corresponding to the written report, source code, and demonstration, which

are listed in the second, third, and fourth columns, respectively. Notice that there seems to have been improvement in the total scores over the semester, which is mainly due to the increase in the demonstration scores. This shows that the performance of the students increased with each assignment, which is an indirect indicator of the sustained motivation of the students. The source codes were of high quality, with adequate comments, sufficient modularity, and meaningful naming of functions and variables. The documentation aspect of the performance, reflected in the reports, was okay but could have been better. This is mainly due to the fact that the students tended to put off writing the report until the last day.

Table 4: Average assignment grades

Assign.	Report (5)	Source (5)	Demo (10)	Total (20)
One	3.14	4.98	6.48	14.96
Two	3.91	5.00	7.64	16.64
Three	3.95	5.00	8.21	19.00

C. Perceptions of the Students

At the end of the semester, we also let the students evaluate the use of the image related materials and assignments. The evaluations were anonymous, however, they were free to identify themselves if they chose to. Table 5 summarizes the evaluation. The average time spent on each assignment was 48.6 hours. Large majority of the student felt that the number of assignments was adequate, as was the image-related support material that was provided. When asked if they would recommend this course to others, the responses were either "simply recommend" or "highly recommend". *No* student indicated that he/she would *not* recommend or even be ambivalent about the recommendation. (It may be noted in this context that the students do usually have a choice of sections, taught by different instructors, each semester. Besides, I do not teach this course every semester.) The students also felt that the assignments "improved significantly" or "greatly improved" their understanding of the basic data structures concepts, programming skills, ADT design skills, and the topics that were subject of the individual assignments. However, they felt that the documentations skill only "somewhat improved" because of the assignments.

The written comments of the students are listed in the Appendix. They reveal the following:

1. Although they considered the assignments challenging they like the idea of image-related assignments, which not only provided them with a real application context but also the ability to visually appreciate the results of their effort.
2. One student, who had previously taken (unsuccessfully) data structures with another instructor, commented that he/she found the image-related assignment to be "effective" and "useful". The previous instructor had used the "game of life" example, which was in the textbook, to illustrate the ADTs. This student did not find that to be particularly useful.
3. Despite the hard work that was demanded by the assignments, the students liked the programming experience and perceived the experience to have great value.

Table 5: Summary of student evaluations of the use of the image related materials and assignments.

Aspect Evaluated	Avg.	Scale
# of hrs spent	48.6	Hours/assignment
# of assignments sufficient?	Yes	Large majority
Was the related materials adequate	Yes	Large majority
Would you recommend the course to others?	1.6	-2: Discourage -1: Lightly discourage 0: Ambivalent +1: simply recommend +2: highly recommend
To what extent did the assignments improve the understanding of:		
Data Structure Concepts	4.5	1: did not help 2: helped a little 3:somewhat improved 4: improved significantly 5: greatly improved
Programming Skill	4.5	
Information Hiding	4.4	
ADT	4.5	
Makefiles	4.1	
Modularity	4.4	
New ADT Design	4.4	
Stacks/Queues	4.1	
ListOfLists	4.5	
Graphs	4.2	
Depth- and Breadth-first searches	4.7	
Image processing Concepts	4.7	
Documentation Skills	3.4	
Importance of code Commenting	3.6	

D. Perceptions of the Instructor

In my experience, the time and effort behind using image-related assignments and examples is not significantly more than taught without it. There is some effort upfront, before the beginning of the semester, in preparing the booklet regarding the image-related background and the assignments. Once prepared these are made available towards the beginning of the semester so that the students have a clear picture of what is expected from them. They are also told that they would have to schedule significant amount of their time for this course, which contrary to expectations, does not deter them from taking the course. I have taught this course in

previous semester using image-related tasks; so most of the students who register have a good idea of what is expected in the course from their peers.

Most of the students complete their assignments on time. For each assignment, only 2 to 3 out of the 24 students did not complete their assignments. There was only one student, with inadequate prerequisite background, who could not complete any of the assignments. I found the students to be more engaged when image-related examples were being worked out to illustrate data structures concepts, than when not using image-related examples.

Another particular advantage of using image-related examples is that they easily provide for many active learning scenarios [13] in the classroom when the student are actively engaged in groups to solve example problems. The visual nature of the outputs allows them to understand each other's lines of thoughts as the solution is being constructed and thus, provide for greater chances of peer based learning based on mutual feedback.

As a secondary effect, students do get drawn into computer vision research at an early stage. One student from the Spring 2001 is presently engaged in computer research in our laboratory and is seriously thinking about graduate school.

IV. Conclusions and Future Directions

In this paper, we have documented our efforts in evaluating the effectiveness of image-related assignments and examples to teach an undergraduate data structures course. This integration of image computations into the undergraduate curriculum at a basic level has many advantages, one of which is that it helps in the understanding of the basic data structures concepts and keeps the students engaged throughout the semester. We have found that the image-related assignments significantly improve the understanding of the data structure concepts.

In future, we plan to pursue evaluations that are aimed at answering questions such as: Is there any difference between students who are taught data structures using image-related assignments and those who are not, in terms of performance in subsequent courses? Given the same instructor, are there perceivable differences in student performances when data structures is taught with image-related examples and when it is not? How effective is it for instructors, who are not experts in computer vision, to teach using image-related examples?

Appendix

A. Student Comments

1. "The assignments were very good. It is much more rewarding to build an useful application over the course of the semester as opposed to bunch of useless little programs. However, the requirements may be a little too stringent. While good documentation is very important, most of the students each have 4 to 5 other classes, which are also quite time consuming. It can be very difficult to accomplish the programming features of this class and create a good report while also maintaining good grades in all the other classes. I think for this level, the focus should be on the application itself and leave the stringent documentation requirement (to) higher level classes where students should have a very form grasp of programming."
2. "Excellent assignments. They tested, infuriated, and impassioned me. I would suggest additional help early and more thorough grading."
3. "This is the best class I have ever taken at USF. I learned more about programming in 3 months than I learned the first portion of CSE curriculum."
4. "It is a nicely put together course. Everything fits together, and I learned a lot."
5. "I found the assignments with image related tasks to be very unique. Writing the actual programs for these assignments was a very tedious task for me. If I were familiar or had a better understanding of image processing, I probably would have done better on the projects that were assigned. It was very different using image related tasks. Another point is that my previous programming instructor was not a "good" instructor compared to you. While doing these assignments, I often had to go back and read over topics that I thought should have been covered in Program Design. Professor Sarkar, my grades do not reflect what I have learned in this course. You are an excellent instructor for this course and I thank you for helping make my education experience meaningful to me."
6. "I came into this course having only Fortran and Program Design as programming experience. I took program design during the summer (2000) in Lakeland and through much hard work and the grace of God, I made an "A" in the course. I signed up for Data Structures in Fall 2000, and flunked. That being said, let me emphasize that there is a SIGNIFICANT difference between the Data Structures Fall 2000 course and the Spring 2001 course. During the Fall course, the material was presented directly from the book. I found the book AND the instructor both confusing and unhelpful. The "game of life" presentation of the material was not effective in teaching structures or the development of ADTs. In contrast, I found the use of the image processing clearly demonstrated the usefulness of the material presented, while also being relevant to today's technology. I can honestly say I've learned more in this course than I have in the last two years of education here at USF."
7. "Was very difficult; however, I feel that I have learned much more than most students have taken Data Structures from other professors. I feel this method of

teaching Data Structures is more interesting for the student, which in turn increases their motivation for learning. You can see the final product after the class is over with not to mention a great program to show future employers as part of your "interview toolkit."

8. "The 3rd assignment was difficult to determine if code was bad or just took a long time; perhaps could use a centroid instead of comparing every space within each region.

ACKNOWLEDGMENT

I would like to thank Dmitry Goldgof and Kevin Bowyer for their feedback regarding the integration process. Yelena Mukomel, who was the TA for the course, also deserves special mention for meticulously grading the assignments and keeping detailed, itemized records of the students' grades.

REFERENCES

[1] Angelo, T. A., and Cross, K. A., "Classroom assessment techniques: A handbook for college teachers," San Francisco: Jossey-Bass, 1993.

[2] Barg, M., Fekete, A., and Greening, T., "Problem-based learning for foundation computer science courses," *Computer Science Education*, vol. 10, no. 2, pp. 109—128, Aug. 2000.

[3] Castleman, K.R., "Digital Image Processing," Prentice Hall, 1996.

[4] Erthal, M. J., "Analysis of performance in microcomputer applications class," *Delta Pi Epsilon Journal*, vol. 40, no. 1, pp. 36—49, Winter 1998.

[5] Geller, J., and Dios R., "A low-tech hands-on approach to teaching sorting algorithms to working students," *Computer Science and Education*, vol. 31, no. 1, pp. 89—103, Aug 1998.

[6] Greenberg, R., "The IPT project: Image processing for teaching," *T. H.E. Journal*, pp. 61--64, Dec. 1996.

[7] Greenberg, R., et al., "Image processing for teaching," *Journal of Science Education and Technology*, vol. 2, pp. 469--480, 1993.

[8] Hancilles, B., and Shankararaman, V., "Multiple representation for understanding data structures," *Computers and Education*, vol. 29, pp. 1—11, Aug 1997.

[9] Horowitz and Sahni, S., "Fundamentals of Data Structures in C," W. H. Freeman Press, 1993.

[10] Jain, R., Kasturi, R., and Schunk, B. G., "Machine Vision," McGraw Hill, 1995.

[11] Mahanti, P. K., Farhan, A. K. N., and Hyassat, K., "Laboratory teaching of data and file structure," *Advances in Modeling and Analysis (A)*, vol. 31, no. 2, pp. 29--35, 1997.

[12] Michail, A., "Teaching binary tree algorithms through visual programming," in Proceedings of the *IEEE Symposium on Visual Languages*, pp. 38--45, 1996.

[13] Myers C., and Jones, T. B., "Promoting Active Learning," San Francisco: Jossey-Bass, 1993.

[14] Raphael, J., and Greenberg, R.. "Image processing: A state-of-the-art way to learn science," *Educational Leadership*, vol. 53, no. 2, pp. 34--37, 1995.

[15] Ritter, G. X., and Wilson, J. N., "Handbook of Computer Vision Algorithms in Image Algebra," CRC Press, 1997.

[16] Rowe, G. W., and Gregor, P., "A computer based learning system for teaching computing: implementation and evaluation," *Computer and Education*, vol. 33, no. 1, pp. 65—76, Aug 1999.

[17] Sarkar, S., and Goldgof, D. G., "Integrating undergraduate level data structures education and image computations," in *IEEE Computer Society Workshop on Undergraduate Education and Image Computation*, 1997.

[18] Sarkar S., and Goldgof, D. G., "Integrating image computations in undergraduate level data structures education," *International Journal on Pattern Recognition and Artificial Intelligence*, vol. 12, no. 8, pp. 1071—1080, 1998.

[19] Shaffer, C. A., Heath, L. S., and Yang, J., "Using the SWAN data structure visualization system for computer science education," *SIGSE Bulletin (ACM Special Interest Group in Computer Science Education)*, pp. 140--144, 1996.

[20] Silverman, R., Welty, W. M. and Lyons, S., "Case studies for teacher problem solving," New York: McGraw Hill, 1992.

[21] Weimer, M. G., "Teaching Large classes well: News Directions for Teaching and Learning," no. 32, San Francisco: Jossey-Bass, 1987.

[22] Whale, G., "DRUIDS: Tools for understanding data structures and algorithms," *Proceedings of the IEEE International Conference on Multimedia Engineering Education*, pp. 403--407, 1994.