Computer Vision Research Teaching Modules for Community College Computer Science and Engineering Courses

Dwight Egbert, George Bebis, and Dave Williams
University of Nevada, Reno and Western Nevada Community College

Abstract

This paper describes computer vision teaching modules we have developed for use in several courses at the University of Nevada as part of a Combined Research and Curriculum Development (CRCD) project, sponsored by NSF. Modules developed to date include programming projects for CS1 and/or CS2 courses, an image compression module for an introductory logic course, a digital camera interfacing module for a microprocessor course, and several programming modules for use in a data structures course. We have also found that these modules can be used effectively at the community college level and can provide resources to community college faculty that they might not otherwise have readily available. In fact, the use of computer graphics and image processing programs as teaching and motivational tools is becoming common at all levels of education. As an example, one of our modules used in CS1 provides a brief background in computer vision concepts and allows students to write an image processing program with applications in computer vision. Using concepts learned in a first programming course students can read in a two dimensional array of data from a file that represents a black and white photographic image, perform one or more transformations on the data, and write the transformed data to a new file. A simple image viewer program can be used to display the before and after images and students can actually see and understand the effects of the transformation. In addition to learning more about the target subject it is the intent of the modules that students also have some fun with images. Many students do indeed enjoy the visual nature of the projects and are surprised that they can accomplish so much in lower division courses. Instructors wishing to include computer vision into their courses can easily modify a given module's contents and adopt all or parts of any given module. The modules are available for free use or adaptation by other instructors and institutions. http://www.cs.unr.edu/CRCD/

Index Terms

Computer Vision, Image Processing, Programming Projects, Teaching Modules

Introduction and Background

Students have long complained, and rightfully so, that traditional programming exercises suitable for introductory computer science and engineering classes are largely uninteresting and uninspiring. Students vastly prefer to write programs which have some perceived value to them or which pertain to their area of study. Regrettably, most programming tasks beyond the mundane are very difficult to implement at the introductory level, so beginning students are relegated to exercises which often provide little motivation to stretch their developing skills. However, computer vision and image processing provide a mechanism for implementing useful

and visually demonstrative programming projects which are well within the scope of beginning programmers. Likewise, as the programming students progress additional processing functions can be built on previous work.

In fact, the use of computer graphics and image processing programs as teaching and motivational tools is becoming common at all levels of education. The motivation for introducing computer vision and image processing into the high school (and even middle and primary school) curriculum is vividly described by Thomas et al [1].

> "Vision is the sense through and by which we perceive and understand our world. … Learned eye-body coordination makes it possible for us to act and/or react smoothly and efficiently in all sorts of vision-guided situations. … It is also a powerful medium for communicating complex scientific ideas, especially those involving scientific processes. … We have never seen a technology so appealing to students of all ages as scientific visualization."

Thomas describes visiting a one-room elementary school in Montana and showing students images of Mars taken from the Viking spacecraft. The students became involved in using the images to answer questions about Mars such as the size of craters and characteristics of other geologic features. He states "Over the next hour the class took first an interest, then ownership, then pride in their investigation of Mars."

Several publications by Greenberg et al spanning almost ten years show a similar experience using image processing for upper elementary and secondary teaching [2] – [3]. Their project, "Image Processing for Teaching" (IPT) started as a response to the 1988 solicitation by NSF for Projects to Promote the Effective Use of Technology in the Teaching of Science and Mathematics. Part of the motivation for the IPT project derives from a 1983 survey of teachers in which nearly 85% of the respondents noted that their preferred style for both teaching and learning was visual.

Likewise, image processing has been used in general engineering education at the college level at several institutions. For example, Shultz describes the use of digital signal processing and image processing research experiences in the undergraduate electrical engineering curriculum [4]. Jankowski also describes the use of *Mathematica* for digital image processing teaching modules in electrical engineering education [5]. Jimenez-Peris et al have described their approach to adding depth to CS1 and CS2 courses through the use of several interactive programming projects including games, image processing, and other applications [6]. Andrews et al have also included image processing projects in CS1 through the use of class libraries which include graphics primitives [7].

These representative examples are a few of the many ways to include visually stimulating projects into the curriculum at a variety of levels. Our project emphasizes computer vision research and the teaching modules are one component which integrates computer vision with several basic topics covered in the computer science and engineering curriculum. Teaching modules consist of Power Point presentations, programming assignments and/or laboratory experiments, and in some cases complete video taped lectures. The modules can be easily

adapted to a variety of courses and integrated into the existing course material. The modules can be adopted as-is or can be used as a resource for custom tailoring specific projects.

The strengths of these teaching modules include 1) a basis of interesting and enjoyable program development topics that appeal to students, 2) a well-integrated foundation of accompanying instructional material to prepare students for the code they are about to write, 3) implementation of the assigned programming tasks in a manner suitable for students with a wide variety of skills, and 4) the ease with which the instructional material and programming tasks may be easily integrated into a wide variety of curricula and modified if desired to best serve the instructor's educational requirements.

Integrating Computer Vision Research into the CS and Engineering Curriculum

Computer vision is an ideal area for integrating research with teaching. As described above, computer vision has an immediate appeal to most students due to their intimate relationship to visual experience and people's fascination with this sense. Students have the opportunity to literally "see" the results of applying theory to solve practical problems. We believe that using computer vision research results can provide a high level of motivation to students. It is also an excellent learning tool for teaching students to integrate and use their acquired knowledge.

When confronted with a programming task that provides a challenge beyond their own expectations and which rises above the typical text-based numerical problem, students become energized and often rise to the occasion. For example, most introductory computer science and engineering students would never expect their skills to develop sufficiently in one semester to permit them to write useful image processing functions. Once they discover this task is well within their reach, their effort and enthusiasm is observed to increase considerably. By combining the necessary theoretical material and instructional tools into an integrated package which is highly flexible for adaptation in a wide variety of programming courses, the integrated computer vision image processing module readily permits the completion of apparently sophisticated programming tasks by students with relatively minimal programming skills.

Our CRCD project objective is to integrate computer vision research seamlessly into and throughout the curriculum and quantify its effectiveness. Our philosophy is that students should be introduced to research as soon as possible in the undergraduate program. In contrast to traditional approaches which tend only to add senior level research courses, our approach for immersing students into research involves the systematic engagement of students into research through well-structured research activities which start from their freshman or sophomore year and continue until their graduation.

In order to introduce students to research as soon as possible, we have integrated computer vision results into traditional "core" courses. Not only does this integrated strategy expose the students to computer vision related research problems at an early stage of the curriculum but it also imparts cohesiveness to the course concepts and brings them closer to real life than is the usual practice. Students are introduced to the learning process by seeking answers to “what if” types of questions from them, encouraging them to predict the outcome of certain experiments and then asking them to explain their results. It should be emphasized that computer vision research results can be naturally integrated in these courses without detracting from the original teaching

goals. We have developed research components suitable for use in each of the following University of Nevada, Reno (UNR) courses:

CS 201 – Introduction to Computer Science I (sophomore level, frequently taken by freshmen)
CS 202 - Introduction to Computer Science II (sophomore level)
CS/EE 236 – Introduction to Computer Engineering (sophomore level)
CS 308 - Data Structures (junior level, portions may apply to soph. specialized courses)
CS/EE 336 - Microprocessor Engineering (junior level, may apply to soph. technology courses)

The integration of computer vision research into these course is being done through self-contained modules in such as way as to make the integration easily transferable to similar courses. The modules include lecture notes and Power Point presentations, example student design projects and labs, recommended reading materials, a prerequisite list, and in some cases video taped lectures. The list of prerequisite knowledge can be used by instructors at other institutions to aid in deciding whether or not the research results are of an appropriate level for their course. They can also use single modules, multiple modules, or modify pieces of individual modules to enhance specific course goals.

Computer Vision in CS1 and CS2

The Introduction to Computer Science (CS 201 and CS 202) courses are typical CSAB CS1 and CS2 courses, the first two programming courses for most computer science and engineering students. The introduction of computer vision research in these courses comes primarily at the end of the semester in CS1 or the beginning in CS2. For example, an image processing final project in CS1 might be introduced by one lecture covering research principles and computer vision basics followed by a second lecture dealing with questions about the project. Usually, the project also brings out questions and discussion during parts of several more class periods as students progress through it. Image data are introduced as an example of two dimensional arrays. Basic concepts such as edges in images being changes in brightness are explained so that students can understand the processing they will perform. The project as implemented at UNR consists of a menu driven image processing program with several functions including file read and write, negative, rotate, threshold, and basic filter. The necessary programming does not involve anything beyond what is normally covered in the course.

At the beginning of the project students are given a function to read a black and white portable gray map (PGM) file and use this as a model to write their own function to write data back to a file. All other functions involve user input/output and simple array manipulations. One of the image processing functions is an operation of the student's choice. Many students were very creative in their choice of operations and most students enjoyed the project. The visual feedback made them feel they could actually accomplish a lot with only one programming course.

While the modules are designed to be used as comprehensive programming exercise and will provide their greatest value in this mode, they also permit a high degree of flexibility for adaptation in lesser degrees of complexity. Once the necessary background material and relatively basic programming tools have been presented to students, they will be fully equipped to write the entire image processing program on their own. However, many (if not most) students

will have the greatest difficulty writing and perfecting the program overhead, particularly file I/O. Given sufficient time and opportunity, constructing the entire program is a very valuable exercise for students that constitutes an excellent final course project. First semester programming students enrolled in other courses will likely require a minimum of several weeks to properly complete a programming project of this magnitude, and additional lecture time beyond the initial module presentations will often be necessary to address the students' questions.

Compared to program overhead, most students will find that the actual image processing functions are considerably more straightforward and rewarding to implement. Some curricula may not present all of the necessary programming material on a schedule which allows sufficient time for the entire project, some may not have enough available lecture time to devote to the full project, and other courses may prefer to focus on algorithm development in lieu of C++ mechanics. In these cases, students may be provided with the module background material in a single class lecture, given a source code framework containing all of the necessary program overhead, and then be tasked to develop just the actual image processing functions on their own. With this approach, students will spend the available time concentrating on the image processing algorithms which lie at the heart of this module. Relieving them of the responsibility to develop the entire program independently does omit the opportunity to further reinforce those necessary and valuable skills. However, in cases where time is short or the curriculum favors algorithm development over perfecting coding skill, the inherent flexibility of the module still permits its effective use in as little as one week's time.

The Introduction to Computer Vision Lecture

During the introduction to computer vision lecture the principles of scientific research are reviewed so that students are reminded of the scientific method before beginning their exploration into image processing and computer vision. The concept of images as two dimensional arrays is discussed at length. The relationship between our visual experience and the array element properties which must relate to that experience are demonstrated. For example, an "edge" in an image is a result of a transition between dark and light pixels along either side of a line. Likewise, thresholding a gray scale image to get a binary black & white image can be used to demonstrate regions and histogram analysis.

An effective introduction to computer vision is to ask students to raise their hands if they recognize the object shown in Figure 1.

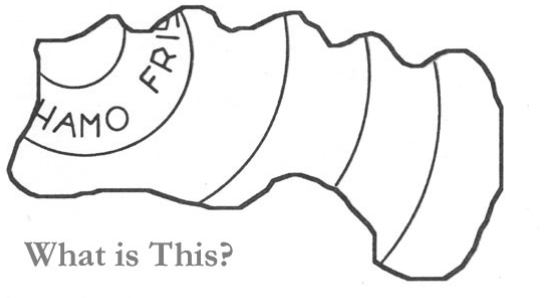

Figure. 1 Image of a Whamo Frisbee used to introduce object recognition. (adapted from [8])

After a few moments the object is identified as a chewed up Frisbee, common enough on college campuses. Then, students are asked to explain how they identified the display as a Frisbee. This first glimpse of the difficulties involved in object recognition and computer vision demonstrates the difference between what we learn to do as humans and what we must do to extract information from arrays of numbers.

After the introduction of object recognition we examine a small portion of the image and compare the visual effect with the data array and the histogram. Figure 2 shows the "M" in Whamo after it is re-sampled to 16 by 18 pixels (for display purposes) together with a histogram of the image. The concept of a histogram and its use in image processing can be explained at this point.

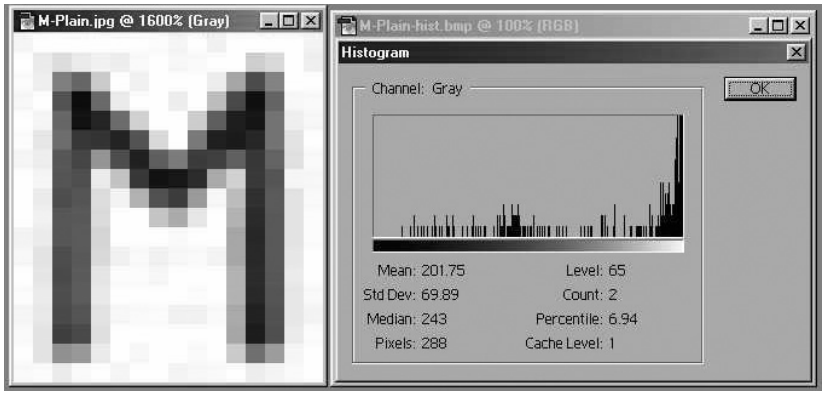

Figure. 2 "M" from Whamo and histogram after re-sampling to 16x18 pixels.

During a brief discussion of image file formats the portable gray map (PGM) format is introduced and two version are demonstrated. The simplest PGM format, the P2 ASCII format, can easily be demonstrated using a word processor. Figure 3 shows the "M" image in PGM P2 format.

```
P2
# Created by IrfanView
16 18
255
254 254 255 255 255 254 255 255 254 254 251 255 255 255 255 254
254 254 235 248 255 255 254 255 254 254 254 255 227 241 255 255
255 255 138 117 213 255 255 254 251 255 255 172 087 158 247 255
255 254 113 039 105 230 255 251 254 255 200 079 039 134 242 255
255 251 125 044 052 133 235 255 255 235 108 049 046 142 243 255
255 254 123 084 081 049 124 243 255 162 052 056 048 133 243 255
255 250 105 120 193 116 046 117 141 069 099 158 082 125 241 255
255 248 102 125 234 210 073 048 051 058 186 210 093 123 241 255
255 246 103 125 225 255 195 095 074 161 255 201 079 113 239 255
255 249 103 125 225 255 255 224 212 246 255 185 073 111 236 255
255 251 106 134 229 255 251 255 255 255 255 180 069 117 239 255
255 247 101 136 230 255 251 254 255 255 255 180 069 120 240 255
255 241 098 135 230 255 254 251 251 254 255 189 072 112 236 255
255 244 101 136 231 255 254 254 254 251 255 191 072 111 236 255
255 246 099 128 226 255 254 254 254 254 255 181 067 110 236 255
255 241 084 120 225 255 254 255 254 254 255 173 051 111 239 255
255 245 160 194 247 255 255 255 255 254 254 221 144 185 250 254
254 254 247 251 255 251 254 255 255 254 254 255 246 250 254 255
```

Figure. 3 "M" from Whamo 16x18 pixel image in PGM P2 format.

The first line contains “P2” to identify the format, subsequent comment lines begin with an ASCII “#” character, the next line contains the number of columns and number of rows of pixels, and the next line contains the maximum gray value. The rest of the file contains the pixel values in row order. The image character of the file display can be detected if the smaller numbers representing dark pixels are somehow identified.

The same image in PGM P5, binary, file format can easily be examined with DOS Debug or other file viewers. Figure 4 shows a display of the hexadecimal and ASCII pixel values for the “M” image file in P5 format.

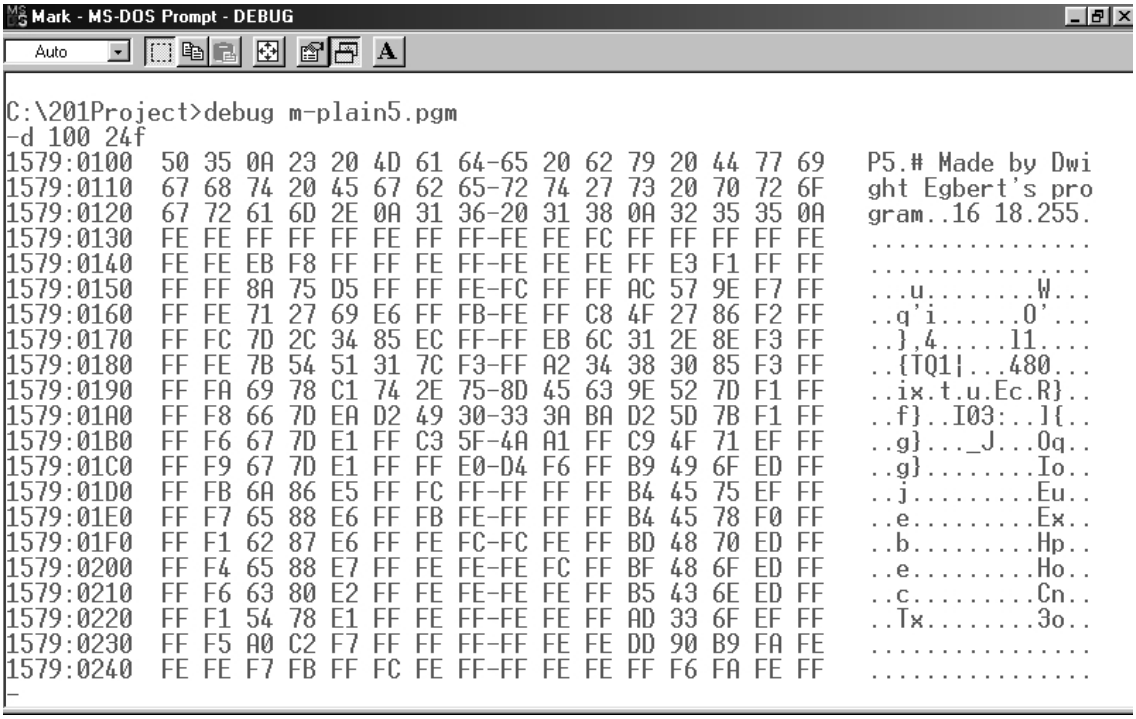

Figure. 4 Debug display of PGM P5 image file format for “M” from Whamo .

Both file displays show the different numeric values of array element for areas within and without the “M” demonstrating the relationship between intensity and array element value. However, it is visually difficult to quickly identify the “M”. This provides a basis for introducing the concept of thresholding images to make edges more detectable. A threshold value to be applied to the image is then chosen based upon the original histogram display. The shape of the histogram is described and the meanings of peaks and valleys are discussed. The program listing shown in Figure 5 demonstrates how simple it is to perform a threshold on a two dimensional array representing an image. The four lines of code in bold are actually all that is required for the thresholding operation. The rest of the function is overhead and I/O. Figure 6 shows the Debug display of the resulting P5 thresholded “M”.

```
int MakeThd(int pixelsTd[][640], int ppLineTd, int numLinesTd, int MaxGLTd, int CurAryTd)
{
 int dmy1, dmy2, dmy3;
 char dmystr[20];
 if(CurAryTd == 0)
 {
  cout << endl;
  cout << "Enter the threshold value (between 0 and 255) >";
  cin >> dmystr;
  dmy3 = atoi(dmystr);
  cout << endl;

  for(dmy1=0; dmy1<numLinesTd; dmy1++)
  for(dmy2=0; dmy2<ppLineTd; dmy2++)
     if (pixelsTd[dmy1][dmy2] < dmy3) pixelsTd[dmy1][dmy2]=0;
       else pixelsTd[dmy1][dmy2]=255;

  cout << "CURRENT IMAGE ARRAY IS THRESHOLDED FROM THE PREVIOUS IMAGE ARRAY." << endl;
  return(0);
 }
```

Figure. 5 Sample C function to perform a threshold of a 2 dimensional image array.

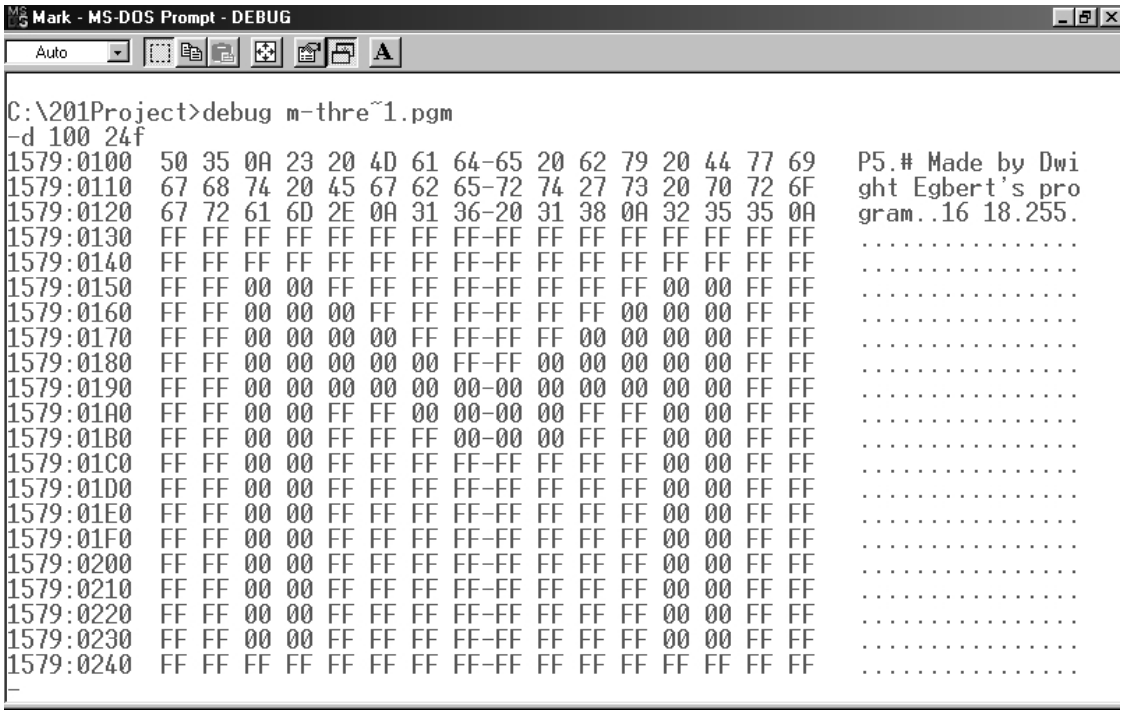

Figure. 6 Debug display of thresholded image file for "M" from Whamo .

The result of the thresholding is now immediately obvious in the array element values which are either 0 or 255 (FF hex). The concept of edge detection is now discussed and the relative difficulties involved with detecting edges on raw as distinct from thresholded images can be demonstrated. Note that on the P5 format image files the length of the single comment line was carefully chosen so that the upper left hand pixel begins on the first column of the 16 column display. This 16 column display is also the reason the "M" image was re-sampled to 16 by 18 pixels. Such formatting considerations are only necessary for creating the teaching illustrations.

These and other concepts of computer vision, object recognition, and image processing are related to the programming principles and constructs which the students have been learning in the CS1 course. The images in particular give meaning to arrays and make the two dimensional arrays come alive. A brief introduction to hexadecimal is also given to discuss the P5 format. However, knowledge of hexadecimal notation is not a prerequisite.

The Programming Project

The objective of our sample project is to write an image processing program with applications in computer vision which will read in a two dimensional array of data from a file that represents a black and white photographic image, perform one or more transformation operations on the data and write the transformed data to a new file.

A sample executable program and image files are available on the laboratory server for students to test. Students can view the image data files both before and after performing transformations on them using a freeware program *IrfanView* [9] which can be downloaded from: http://www.irfanview.com. *IrfanView* also allows students to read in images in a variety of different formats and save them in the PGM format used for the project. An additional advantage of *IrfanView* is that it provides a hexadecimal display of the image file contents. This allows students to examine their image files for correctness and relate image contents to the hex displays used in lecture. An example hex dump from *IrfanView* is shown in Figure 7. The simple image header format for PGM can be read from the first few bytes. For example the basic code for PGM binary files is P5 linefeed (ASCII values 50, 35, 0A in hex) followed by, comment lines which start with # (ASCII 23 hex), image size parameters and finally by raw image data.

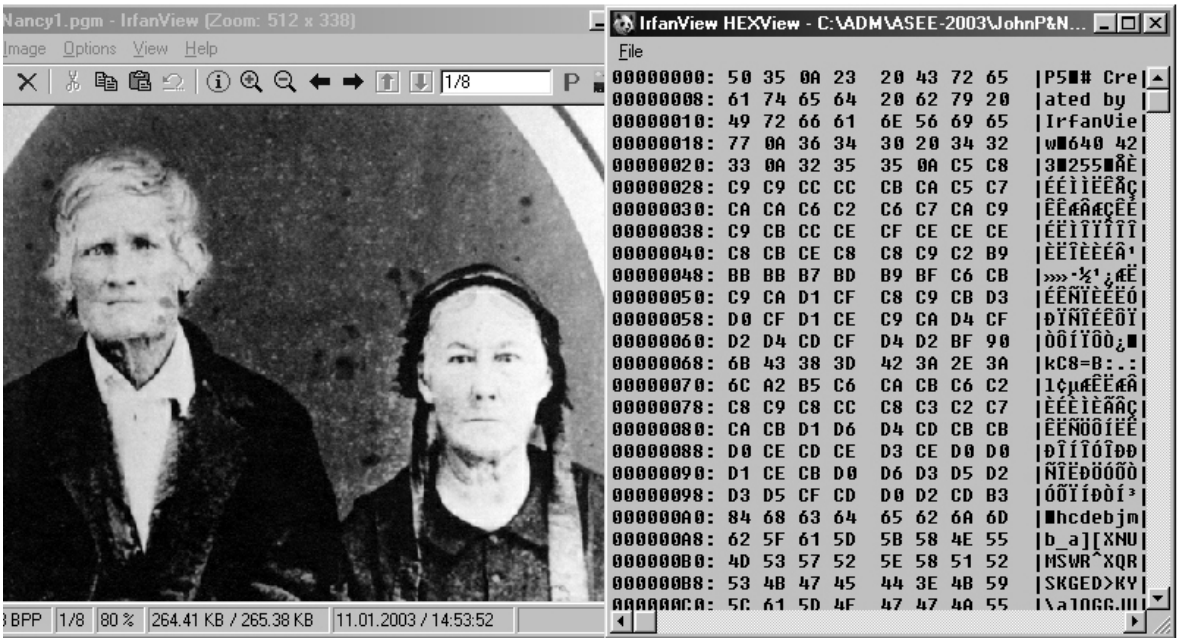

Figure. 7 IRFANVIEW Hex display for sample PGM P5 Image .

As mentioned previously, one of the goals of the computer vision modules is to have some fun and add excitement to the learning process. In this spirit, the author has chosen for this paper as an example image an ancestral photo circa 1870 from Murfreesboro, Tennessee which is about 40 miles from the conference. For their programming projects, students may use one or more of several provided sample image files to test their program, they may use one of the available digital cameras during a lab period to go out and take their own photos, or they may bring in one of their own photographic prints, slides or negatives to be scanned and converted to a data file.

Students are provided with the specifications for the PGM file format and a sample C++ function which will read such a file into a two dimensional array. The maximum array size is fixed and the read function has been written so that it does not contain any programming constructs which have not been covered in the CS1 course. The project is written in standard C++ and does not require special Windows functions. In our course we use Visual C++ and the laboratory machines use thin client stations with the server running Windows 2000. However, the programming project could just as easily be implemented on UNIX or LINUX systems. *IrfanView* is a Windows image viewer that is very easy and fast to use. However, GIMP or any one of several other viewers could be used. Each semester the project transformations, format and design can be varied. During the first semester we used this project we defined ten transformations and functions which each student must implement using a menu structure in their main program. An example menu listing of the ten operations is shown in Figure 8.

```
Welcome to My Computer Vision Software (rev. 1.5)

Please Select the Operation You Want to Perform.

1.  Load current image array from file.
2.  Reload current image array from same file.
3.  Write current image array to file.
4.  Create a negative of the current image array.
5.  Specify filter array contents.
6.  Filter current image array.
7.  Threshold current image array.
8.  Rotate current image array 90 deg. clockwise.
9.  My own transformation of current image array.
10. Exit program.

Enter Menu Item >
```

Figure. 8 Example Main Menu Display for Student programming project .

All image transformations are performed on a "current image array" so that multiple transformations can be accommodated. This is consistent with many graphics and CAD systems which use a "current position" and/or a "current object". An example screen display is shown in Figure 9 which shows the main menu running in a DOS window together with three different images being displayed by three different instances of *IrfanView*. The top left image in Figure 9 is the original file, the bottom left is a thresholded image and the bottom right is a negative image that has also been flipped horizontally.

This is representative of the kind of user interface that can be achieved with readily available image viewer programs today. This does not impose the burden of direct Windows or graphics programming on students in a first programming course. Instead, they can concentrate on the programming principles they have been studying during the semester and still have the advantages of dynamic and fun image displays. They can get direct visual feedback showing the effects of their programming functions.

The programming project described here can provide the basis for many different image manipulation functions and transformations. Successful coding of the more elementary processing tasks is usually within the grasp of even the most reluctant programmers. For example, the concept of thresholding is easily explained and understood, and its proper implementation in C++ (as presented above) follows directly from this understanding. Other processing functions are no more difficult to comprehend but slightly more difficult to perform. For example, cropping an image requires the manipulation of output array dimensions (as does rotating a non-square image). Once the student is able to successfully crop one side of an image, writing a single function which permits simultaneous cropping of any or all of the four sides of the image simultaneously is an exercise which parallels the process of progressive enhancement usually practiced with real-world code development. Students should be encouraged to begin with processing tasks well within their current abilities and to further hone their skills by subsequently tackling the more advanced aspects of the module.

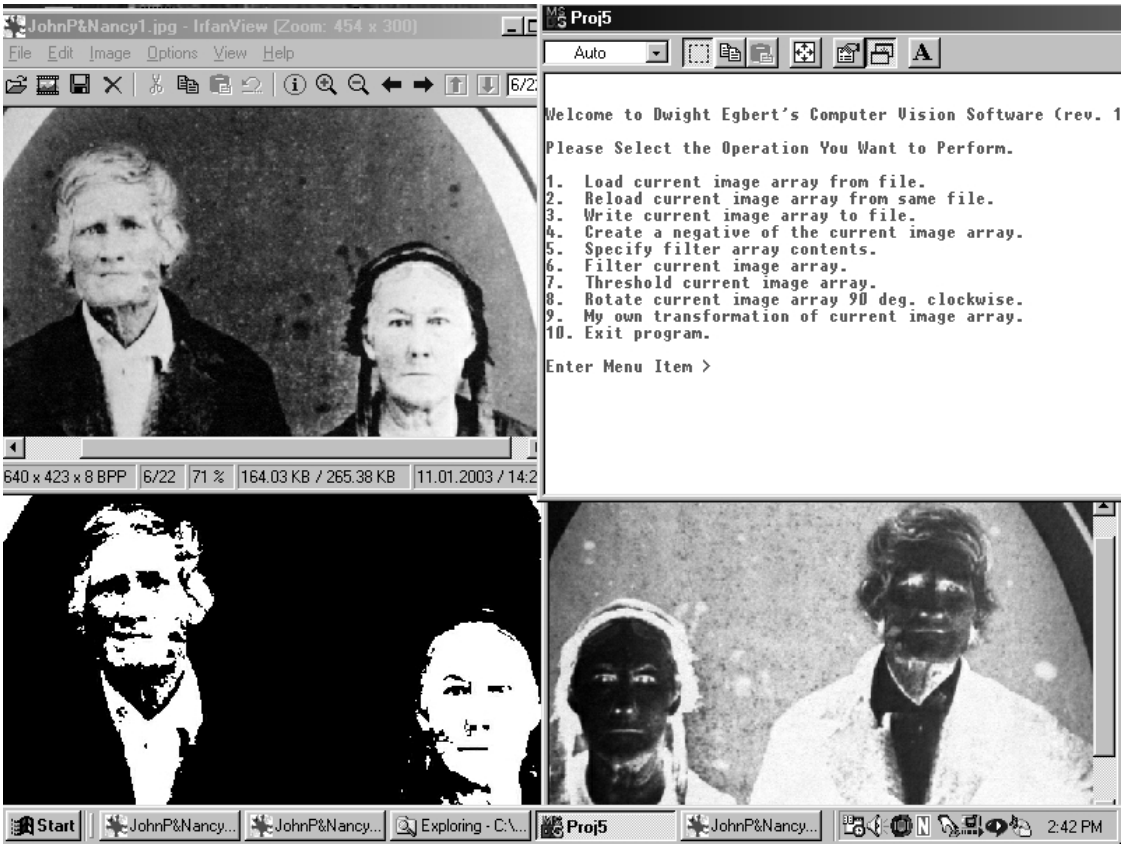

Figure. 9 Example Screen Display for Student programming project .

Within each module, students with rapidly maturing abilities will have an abundant opportunity to pursue significantly advanced processing tasks or user interface development. In fact, at least one student was able to use the image displays to find and correct a programming bug which caused the output images to be distorted with a specific pattern. Certain functions will require more creativity to achieve than will others. Perfecting an algorithm which gradually fades an image from normal contrast at one edge of an image to a solid shade (white, gray, or black) at the opposite edge will likely require considerable thought and algorithm refinement from most students before a sufficiently smooth gradation is achieved. Similarly, correctly pixellating an image (changing the shade of an entire block of pixels to a single value representative of the entire block) requires consideration of many ancillary factors beyond the basic mechanics required of the code (which is not an easy task by itself). Left to their own devices, some students may settle on using the mean value or median value of the pixels within each block while others may devise a more complicated algorithm. Without understanding the reasons why, students will be able to differentiate between acceptable and unacceptable results and will be driven to experiment with various possible solutions before settling on the version which produces the best results in their estimation. This is an excellent illustration of the advantages of results presented in an intuitive, visual format. Extensive numerical verification of program results is not required; the image either looks right, or it doesn't.

With even minimal thought, a large number of additional processing tasks may easily be realized. The module presents a representative set of possibilities, but individual instructors are free to augment those provided with others of their own creation. Students, especially those with more developed coding abilities or imaginations, should be encouraged to devise their own. The intuitively simple concept of modifying visual images and the almost limitless number of ways in which they may be modified will encourage students to combine their creativity and blossoming programming skills as they would during actual program development. In order to achieve a particularly interesting effect, students may discover the need to learn additional C++ language skills. Encouraging them to take this step on their own will demonstrate the unbounded nature of computer programming. In essence, there are only two levels of programming: that which has already been done, and that which is awaiting implementation by someone with sufficient creativity.

While they are perfectly suitable for use in traditional curricula, the computer vision modules also lend themselves perfectly to alternative instructional techniques. Many programming classes consist of lecture and independent student development of code based on the concepts and practices presented in lecture. As some students may not respond particularly well to this approach, the modules have been designed to permit their presentation in alternative instructional formats.

As presented above, the visual results provided by this particular module are simple to evaluate. Allowing students to compare the methodology, operation, and visual results of their own algorithms to the anonymous work of other students has proven to be very beneficial to all. All traces of authorship should be stripped from student programs before they are made available to the entire class. Students may then be asked to evaluate and comment upon each program. Even those students with lesser skills are able to quickly identify bad attributes and incorrect results of most programs, while almost all students can immediately identify desirable features or

techniques which they may quickly add to their own repertoire. This also provides the students with some perspective and appreciation for the instructor's burden when grading their programming assignments. As anyone who has done so can testify, evaluating code written by students can be an adventure, and sharing an appreciation of that adventure with those responsible often leads to some measure of student improvement.

Similarly, students having difficulty with programming concepts often find that working cooperatively in a group with other students is sufficient to get them back on track. Working in a lab environment, groups of students may be challenged to develop the full slate of image processing functions in increasing order of complexity. Such groups should be carefully chosen to accomplish the desired objectives. Students of dissimilar abilities may be grouped together in cases where the more advanced students are willing and able to mentor those in need of assistance. Allowing the stronger students to complete the work while the weaker students merely observe will not serve the latter group well. In those cases, groups should probably be composed of students of comparable abilities. The broad range of available image processing functions will provide an entry point to the exercise and material of increasing complexity which is suitable for the entire range of student abilities.

Alternatively, students can write stand-alone functions or instructors can provide other frameworks, in their own programming style, for students to expand upon. This permits students to focus on algorithm development in cases where time constraints prohibit completion of the entire program by students or when algorithm concepts are deemed to be more important than generating the program overhead.

Conclusions

Many beginning computer science and engineering students find the task of writing their first complete programs to be quite daunting. Some students will even confuse themselves into thinking that they don't know where to begin a program of their own. Others will encounter seemingly insurmountable obstacles (fundamental programming errors or perplexing syntax errors) which may seriously diminish confidence in their budding ability and which may even be sufficient to cause them to drop the course. In these cases, multiple tasks with an escalating level of difficulty all contained within a single captivating programming exercise may prove beneficial to instill the confidence necessary to gain mastery of the material. As the actual image processing functions vary in complexity from exceedingly simple to highly sophisticated, students who begin with easier functions gain the programming skill and confidence necessary to tackle the more difficult functions, all within the same application. This obviates the need to begin an entirely new programming exercise to increase the level of sophistication or to introduce new programming concepts.

At this point we are pleased with the integration of computer vision research into our own core courses even though we will continue to make improvements. An integral part of this program is the assessment of results. One of several instruments we use is a survey of students in each course after the computer vision modules have been completed. The survey results for two semesters in our own CS1 course are discussed in a previous paper [10]. The survey results show

that most students thought the computer vision module helped them with the basic course content which was one of our main goals. Future plans also include working more with community college instructors and assessing their requirements.

Computer vision systems are already becoming commonplace, and vision technology will soon be applied across a broad range of business and consumer products. This means that there will be strong industry demand for computer vision scientists and engineers, for people who understand computer vision technology and know how to apply it in real-world problems. As a result of our integrating computer vision research experiences throughout our curriculum, many students may consider pursuing careers in computer vision. Likewise, the use of the computer vision modules by community colleges may encourage some students to continue their education to complete a baccalaureate or graduate degree.

For the majority of the students who will not pursue a career in computer vision, such a background will prove helpful in other areas such as pattern recognition, graphics, robotics, multimedia, virtual reality and medical imaging. All computer programming students will benefit from the fundamental skill required of the craft: the translation of a given problem into a properly constructed and tested algorithm which provides the desired results. This invaluable skill is lacking in many new students. The computer vision modules provide a level of curriculum enhancement which increases the students' interest, participation, and achievement in this important educational process, all presented within the framework of an enjoyable programming topic which many of them would have scarcely believed to be within their present abilities. We have also demonstrated that even the first programming course can support the introduction of computer vision research in a way that is meaningful to the course content. Computer vision and image processing provide valuable and interesting implementations for demonstrating basic programming concepts.

Teaching modules and support material can be downloaded from links found at our project website: http:// www.cs.unr.edu/CRCD/.

Acknowledgments

The work reported here has been funded in part by a grant from the National Science Foundation, Combined Research and Curriculum Development, (#0088086).

Bibliography

[1] Thomas, D. A., K. Johnson, and S. Stevenson, "Integrated Mathematics, Science, and Technology: an Introduction to Scientific Visualization", *Journal of Computers in Mathematics and Science Teaching*, Vol. 15, No. 3, 1996, 267-94.

[2] Greenberg, R., R Kolvoord, M. Magisos, R. Strom, and S. Croft, "Image Processing for Teaching", *Journal of Science Education and Teaching*, Vol. 2, No. 3, 1993, 469-80.

[3] Greenburg, R., "Image Processing for Teaching: Transforming a Scientific Research Tool Into an Educational Technology", *Journal of Computers in Mathematics and Science Teaching*, Vol. 17, No. 2, 1998, 149-60.

[4] Shultz, R. R., "Experience in the Integration of Digital Signal and Image Processing Research into the Undergraduate Electrical Engineering Curriculum", *Proceedings of the XXXX American Society for Engineering Education Annual Conference & Exposition*, Session 2632.

[5] Jankowski, M., "New Courseware Modules and Software for Digital Image Processing", *Proceedings of the 2001 American Society for Engineering Education Annual Conference & Exposition*, Session 1320.

[6] Jimenez-Peris, R., S. Khuri, and M. Patino-Martinez, "Adding Breadth to CS1 and CS2 Courses Through Visual and Interactive Programming Projects", *Proceedings of the 1999 ACM Special Interest Group for Computer Science Education*, 252-56.

[7] Andrews, P., D. Broline, W. Slough, and N. Van Cleave, "A Set of CS1 Labs Utilizing Graphical Objects and Inheritance", *Proceedings of the 2001 ASEE/IEEE Frontiers in Education Conference*, T3C-10-14.

[8] Hecht-Nielson, R., *Course Notes: Hecht-Nielson Neurocomputer Application Course*, San Diego, CA, January 1988.

[9] Skiljan, Irfan, *IRFANVIEW Freeware Image Viewer*, http:// www.irfanview.com, (last accessed May 24, 2001).

[10] Egbert, D., G. Bebis, M. McIntosh, N. LaTourrette, and A. Mitra, "Computer Vision Research as a Teaching Tool in CS1", *Proceedings ASEE/IEEE Frontiers in Education Conference*, November 6-9, 2002.

Biographical Information

Dwight Egbert received his Ph.D. in Electrical Engineering from the University of Kansas in 1976. He is in his seventeenth year as a full-time academic engineer with the University of Nevada, Reno College of Engineering. He is currently the Director of Computer Engineering Undergraduate Programs and Director of the College of Engineering Computer Center. He has authored more than 80 original research papers and reports dealing with computer applications in computer vision, remote sensing, image processing, and neurocomputing.

George Bebis completed his Ph.D. degree in Electrical and Computer Engineering in 1996 at the University of Central Florida, Orlando, FL. From 1996 to 1997 he was a visiting assistant professor at the University of Missouri-St. Louis. During the summer of 1998, he was a summer faculty visitor at Lawrence Livermore National Laboratory (CASC). Currently, he is an associate professor in the Computer Science Department at UNR and the founder-director of the UNR Computer Vision Laboratory.

Dave Williams is the Engineering Instructor at Western Nevada Community College in Carson City, Nevada. He is a Registered Professional Engineer and has practiced as a consulting engineer for 30 years in the field of antenna design and analysis, computer simulation and modeling, and electromagnetic compatibility, and holds the Ph.D. degree in Electrical Engineering from the University of Nevada, Reno.