Image Processing Programming Projects in an Upper Division Algorithms Course

Karen T. Sutherland
Department of Computer Science
Augsburg College
Minneapolis, MN 55454
suther@navigation.augsburg.edu

Abstract

We introduced digital image processing into our upper division algorithms course three years ago. This course requires a heavy programming component. To satisfy that requirement, we often found ourselves assigning programming projects that consisted of writing code as a direct implementation of a given algorithm, essentially doing what had already been shown in the text. The students learned how to implement the algorithms, but they did not learn much about why they might find these algorithms useful in their careers as computer scientists. Our dissatisfaction with the existing practice, the fact that we were not taking advantage of the capabilities of the Java language and a desire to incorporate new topics into the course motivated our change. Our goal was to assign programming projects which required implementation of the algorithms studied while at the same time providing the students with interesting real world projects. This paper describes our motivation, the projects assigned and the results.

1 Introduction

We have been assigning image processing programming projects in our algorithms course for the past three years. This course is the fourth course in the major, following Introduction to Computer Science, Programming I, and Data Structures. Students usually take the course in the second semester of their sophomore year. At this point in time, the approach has been used for five sections of the course with approximately 70 students overall. We have used the Corman text[2] all three years. There are four to five programming projects assigned during the semester. All are written in Java. We are currently using the Java 2 Platform, Standard Edition, version 1.4.1.

This change in our curriculum was motivated by three issues.

- We did not view programming projects that consisted of writing code as a direct implementation of a given algorithm to be very interesting or worthwhile.
- We were not taking advantage of the graphical capabilities of the Java language. In fact, we were not taking advantage of many of the object oriented properties of the Java language.
- We wanted to address some of the new core curriculum requirements as outlined in CC2001[5], such as event-driven programming, handling runtime exceptions and Human Computer Interaction (HCI). This course seemed like an obvious place to do so.

Image processing applications address all three issues. We are able to assign programming projects which require implementation of the algorithms studied while at the same time providing the students with interesting real world projects. They become comfortable using Java graphics. It's difficult to do such a project while avoiding object oriented programming techniques. Since image modifications or creations are initiated by user actions in a graphical user interface (GUI), the students learn how to build a simple GUI. They also become experienced with event-driven programming and exception handling.

This is not at all a new idea but rather a summary of what we have tried in our particular program. Indeed, Fink and Heath[3] have done image processing in their algorithms course for some time. Stevenson[6] has discussed the use of image related applications in his algorithms course. A comprehensive review of computer vision education at both the graduate and undergraduate level can be found in [1]. This includes image processing as well as use of applications that create images. At MICS '03, Kenny Hunt presented his work using image processing in CS1 and CS2 [4]. He created a class, EasyBufferedImage, so that beginning students could handle the image processing more easily. Due to the fact that this is an upper division course, we do not provide classes for such interactions, but do hand out examples of reading and writing image files, pixel manipulation and creating GUI's.

2 Examples of Assignments

All assignments corresponded to topics covered in class. All required an analysis of time complexity of algorithms used. All were geared to process so much data that empirical time complexity was a real issue. All required submission of source code for a Java graphical application plus a typed report describing algorithms, classes, methods and objects plus problems encountered and how they were solved.

Array Manipulations:
Since some of the students had never done Java graphics, the first assignment was chosen so that it used data structures which the students knew well (one and two dimensional arrays). They could then focus on the GUI and image related issues. They built a Java application which displayed an image and allowed the user to modify that image. Modifications differed each term. Gradually blackening the image, turning it upside down, and switching the red and blue pixel values are a sample of these operations. They were required to move the image into a 2D array, do the modification then create a new image, which required transferring the pixel values to a 1D array.

Sorting Algorithms:
One "sorting" assignment was to apply sorting algorithms to implement median filtering, a common technique for smoothing a noisy image. They were to do the sorting with three different sorts and show the comparison of actual run times with a line graph. An example of the output from this assignment is in Figure 1.

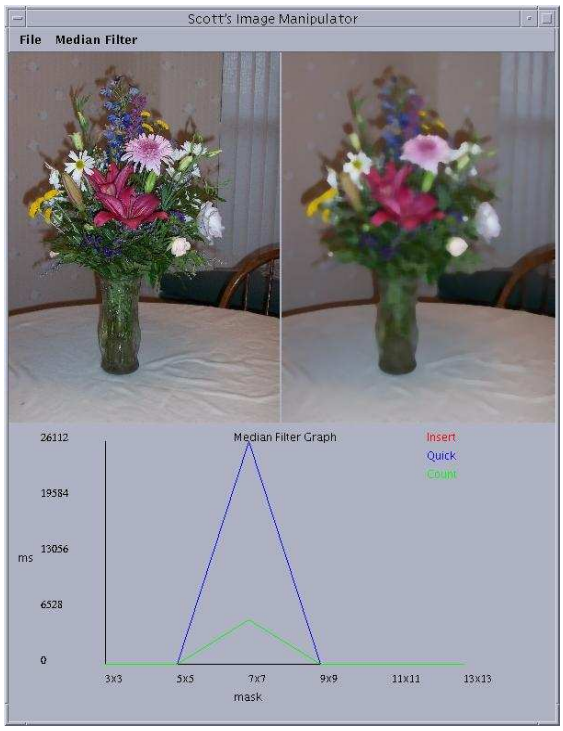

Figure 1: Comparison of sorting algorithms on median filtering used for image blurring. Courtesy of Scott Kuhl.)

In line with the focus of Stevenson[6], a second sort based assignment created an image rather than processing an existing one. It used the Quick Hull algorithm to determine the convex hull of a randomly graphed set of points. Since Quick Hull shares significant similarities with Quick Sort, the students had the same experience of applying recursion that they would have had with Quick Sort but with a more unique application. One of the Quick Hull outcomes can be seen in Figure 2.

Search Tree Algorithms:
Rather than implementing algorithms that used binary trees, we extended our data structure to a quadtree, often used to represent images in different resolutions as well as for image compression. The Java application took an image file with its name input at run time, and averaged pixel values, with a pixel value representing the average of all of the pixels in the image at the root, four values

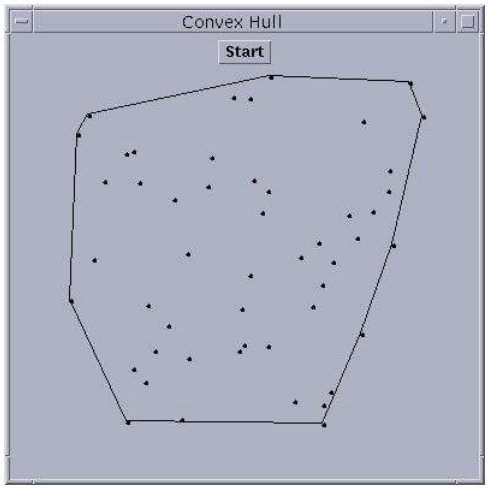

Figure 2: One output showing results of the Quick Hull algorithm implementation. (Courtesy of Ryan Skar.)

representing the average of the values in each quadrant at the next level, etc. A graphical user interface allowed the user to move between levels of resolution, at which point, the image at the chosen resolution would be displayed. The time complexity of the algorithms used to build the tree, store the data and render the image were particularly interesting here. The students were required to think about time complexity and to defend the time complexity of their algorithms as part of the assignment.

Graph Algorithms:
When we covered graphs, they found the number of stars in an image of the night sky by representing pixels in the image above a given "light" threshold with an adjacency list or matrix. They then counted the stars by determining the number of connected components in the graph. This required the use of the standard algorithm for finding connected components. A threshold was set as a program constant. Some of the students set different thresholds to distinguish between different colored stars.

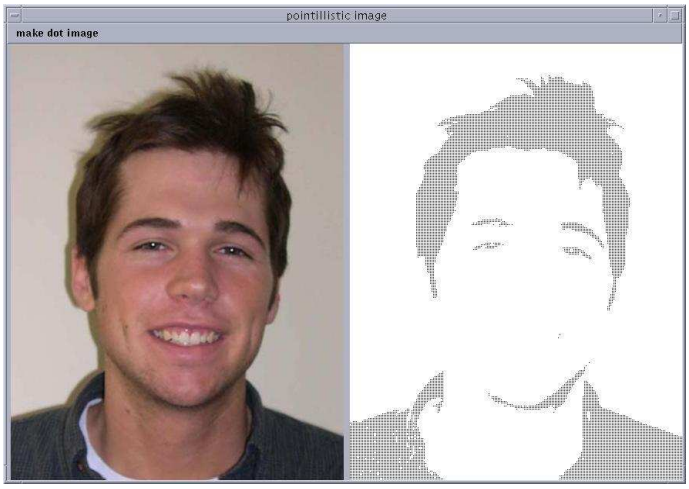

Figure 3: A sample stippled or pointillistic image. (Courtesy of Ryan Bosshart.)

Greedy Algorithms:
When greedy algorithms were introduced, a popular assignment was to make a stippled or pointillistic image of any given image. This project required a defense of why their chosen algorithm was greedy. Some used standard greedy algorithms, some created their own. We then took photos of everyone and ran each student's photo through their own algorithm. Since their individual algorithms were all greedy but not all identical, the results varied significantly. These were posted on the department bulletin board. A sample image and pointillistic result can be seen in Figure 3.

Compression Algorithms:
Huffman coding was implemented by using it to compress an image and then comparing the results to other compressions such as Unix or Run Line Encoding. This gave us a good opportunity to discuss compression algorithms using large files, something which had eluded the author in the past.

Sample Last Assignment:
We try to end each semester with something new and different not necessarily an implementation of a standard algorithm. This year, we are writing a java graphical application which takes a given image and either encodes in the image itself an image signature/watermark or decodes an already encoded signature. We used a simple encoding algorithm. The low level bits of each of the first 10 pixels comprise a binary number which is the seed for the Java Random method. (Note that this is not Math.random, but the Random class constructor.) This can range from 0 through 1023. The low level bits of the next six pixels make up a binary number which is the number of characters in the message (0 through 63). Values are generated and used to represent positions in the 2D image array. Seven positions are required for each ASCII character representation. To encode, the user specifies the message to encode, the seed is chosen and encoded in the image, the character count is taken and encoded, then each character's ASCII code is encoded at the position determined by the seed generated random values. The resulting image is saved. To decode, the seed is decoded from the first 10 pixels in the image, the number of characters is computed from the next six pixels, the bits for each ASCII code are found by going to the positions determined from the randomly generated values. The result is the image displayed with the encoded message printed across the face of the image. One result can be seen in Figure 4.

Figure 4: Decoded output from a "signed" image. (Photo property of the author.)

3 Summary

Now wrapping up its third year, we have tentatively declared the project to be a success. Although initial student reactions have often been hesitant, end of course feedback has been generally positive. At the same time, it has been clear that the strong students were more satisfied than those who were not so strong. However, this has been generally true regardless of how the course is taught.

Overall programming skills, not only those related to processing images, have improved. Every member of our programming team, chosen competitively, has been through this course. Their increased ability to create good data structures, so important in image processing, has given them a strong foundation for tackling many other types of problems. Having this experience on their resumes has served a number of them well when applying for internships and summer research positions. The course has also become much more fun to teach.

References

[1] George Bebis, Dwight Egbert, and Mubarak Shah. Review of Computer Vision Education. *IEEE Transactions on Education*, 46(1), 2003.

[2] Thomas Corman, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw Hill, 2 edition, 2001.

[3] Eugene Fink and Michael Heath. Image-Processing Projects for an Algorithms Course. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(5):859–868, 2001.

[4] Kenny Hunt. Digital Image Processing with Java in Undergraduate Computer Science Curriculum. In *Midwest Instruction and Computing Smposium, 2003 Proceedings*, 2003.

[5] The Joint Task Force on Computing Curricula. Computing Curricula 2001 - Computer Science. IEEE Computing Society and Association for Computing Machinery, December 2001. Available at: http://www.computer.org/education/cc2001.

[6] Daniel E. Stevenson. Image Related Applications for a Core Algorithms Course. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(5):859–868, 2001.