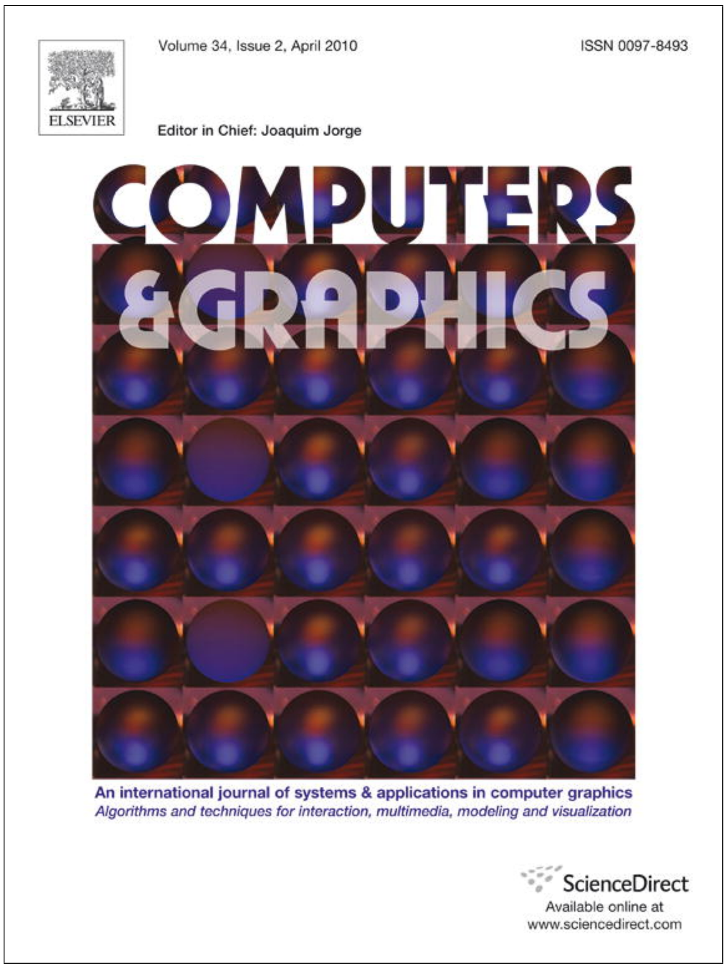
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.


Volume 34, Issue 2, April 2010
ISSN 0097-8493
ELSEVIER
Editor in Chief: Joaquim Jorge
COMPUTERS &GRAPHICS
An international journal of systems & applications in computer graphics
Algorithms and techniques for interaction, multimedia, modeling and visualization
ScienceDirect
Available online at www.sciencedirect.com

(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

Computers & Graphics 36 (2012) 170–177

Contents lists available at SciVerse ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

High-level application development for non-computer science majors using image processing

Amit Shesh

School of Information Technology, Illinois State University, Normal, IL, USA

ARTICLE INFO

Article history:
Received 4 September 2011
Received in revised form
10 January 2012
Accepted 12 January 2012
Available online 26 January 2012

Keywords:
Image processing
Information systems
Computer science education
Introductory programming

ABSTRACT

It is a unique challenge to teach programming and application development to students pursuing an IT degree other than computer science. Using simple visual computing as a medium to teach programming can be very helpful in such situations as it enables programmes that produce pictures rather than raw text and data. This paper describes a semester-long experience of using image processing as the theme in a course to teach programming and program design to students of information systems. Students progressively built a fairly complete image processing application from scratch in a bottom-up fashion using Java. They first concentrated on using low-level constructs like arrays and implemented several operations on them, and then supplemented their programs with features like a GUI complete with “undo–redo” features and capabilities to handle most standard image file formats. Not only did this approach satisfy all the objectives of a typical programming course but also enabled students to develop meaningful applications from scratch with “standard” features. Our classroom was composed of a mix of undergraduate and graduate students lacking sufficient programming background. A comparative analysis shows improvement in student performance when using image processing rather than not. With minor variations, our approach can be fit to courses for other majors where programming is considered useful but not critical.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

Computer programming is considered a basic and vital skill for any student pursuing a career in information technology. While computer science (CS) curricula provide rigorous training in computer programming, other IT-related majors such as information systems (IS) place, at best, a secondary importance to programming. Teaching programming courses in such majors poses several challenges.

First is the incorrect perception among students that computer science is the only IT degree where a knowledge of programming is needed. At our university we see many students choose a non-computer-science degree programme simply due to fear of programming. Many students of such majors mistakenly view programming courses as something they are “forced to take” and something they must “survive”. Compounding this problem is the wide scope of these courses because such majors have fewer programming courses (usually 2) than computer science and yet include advanced concepts like user interface design. Also, majors like information systems emphasize the higher-level aspects of application development and management (i.e. the “bird’s eye” view) as such degrees usually lead to roles of project management and systems analysis in jobs. This often adds fuel to the perception that knowledge of programming is neither relevant nor important to succeed. This behooves instructors to motivate students to work hard and learn computer programming and its benefits without portraying it as an absolutely vital skill as it is for computer science.

Secondly, many such courses have a great diversity in the technical backgrounds of students. This is in part because an increasing number of universities offer courses that are cross-listed across different majors. This compounds the challenge of making course material interesting and comprehensible to all. A third challenge (relevant to computer science courses also) is presented by the ubiquity of commodity computing devices. Most students today are regular users of technology in the form of computers and hand-held devices (for many their interest in IT stems from having used computers before). Thus they expect any programming course to teach them to develop sophisticated applications that they and others can immediately use. However most programming courses result in programs that appear relatively mundane and without practical use, thereby negatively affecting student motivation. Thus pedagogical techniques that help create programs that seem immediately usable and combine concepts of basic programming with high-level application development are highly desirable. This paper discusses the experience of using image processing (henceforth abbreviated as I-P) as a theme to address the above challenges.

E-mail address: ashesh@ilstu.edu

0097-8493/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2012.01.007

Author's personal copy

Visual computing in general is often used as a pedagogical tool because of its illustrative appeal. However, understanding practical forms of multimedia data and operations on them requires knowledge of diverse technical concepts that students of elementary programming courses simply do not have. Although this can be addressed by providing supporting code that simplifies and hides technical details, doing so may diverge from the original objectives of the course. Moreover we contend that writing applications from scratch provides students with a comprehensive understanding of how they work. Therefore we work with simple static images and design projects that provide a balance between simplicity and practicality. Images are inherently interesting to most students because most have stored, classified and annotated digital photographs using commercial software like Picasa [1] and Flickr [2]. Secondly writing programs that work on images provides an emphatic visual feedback unlike staring at raw numbers, and thus likely provides a greater sense of achievement. Technically, images provide natural examples of arrays and their manipulation. Thus programming with images can be simultaneously characterized as "fun" and "technically challenging".

Although images have been used often as examples to teach various concepts in computer science [3–10], they have been largely used to teach only low-level, piece-wise programming constructs. We contend that image processing offers an attractive way to introduce not only basic programming but also design concepts which are of more interest to majors like information systems. Many features present in standard applications, when mapped specifically to an image-manipulation program, pose interesting problems that can lead to a better understanding of design and implementation issues. For example, an "undo–redo" feature is standard in almost all computer applications. How does this feature work in programs like Photoshop? Is it simply a standard design that can be replicated everywhere (and if so, what is it?), or are there important operation and implementation-specific issues?

A primary challenge associated with visual media processing is that it tends to be very mathematical. Many non-CS curricula do not include or place less emphasis on data structures, algorithms and mathematics. Introducing mathematics just to use image processing in a primarily programming course for non-CS majors can prove to be counter-productive to the course goals. We believe it is possible to teach image processing in a strictly "applied" way, exposing students only to implementation details of sophisticated algorithms rather than their underlying theory. Our experience shows some promise in this direction, as students were motivated to complete their assignments in many cases with little idea about how the underlying algorithms worked. Many information analysts go on to design and maintain software systems without acquiring expertise in the domains within which these systems function. We feel our approach assumes relevance in this aspect. Nevertheless we used 1:1 teaching and classroom discussions to encourage those students who were more interested in learning the underlying algorithms.

This paper describes a semester-long experience of using image processing as the theme in a course to teach programming and program design to students of information systems. An abridged version of this experience has been published previously [11]. This paper discusses the motivation and experience in more detail, includes evaluation regarding its impact on student learning and also discusses how our approach can be repeated in similar courses. Section 2 puts our work in the context of relevant previous work. Section 3 discusses the details of our assignments and their objectives. Section 4 discusses a comparative and qualitative evaluation of our work, and finally Section 5 provides a retrospective look at the course, discusses issues to be considered when repeating our approach in similar courses, and identifies avenues for future work.

2. Related work

Visual computing in the form of computer graphics [5] or image processing [12] has been used in programming courses, but primarily in computer science where the expected knowledge of maths and algorithms is higher. Leutenegger et al. [13] use games as tools to teach programming for computer science students, but using multimedia-focused languages. Duchowski et al. [6] use fairly mathematical computer graphics algorithms to teach an advanced programming course. Jordi et al. [14] discuss teaching computer graphics specifically related to information management, thus making it applicable to information systems students. However their goal is to make computer graphics relevant to information systems and not basic programming, and their course also targets computer science students. Guzdial [15] designed a course for non-CS majors that focuses on multimedia computation. Although the foundation of much work in this area, their course uses significant existing material for students to use and extend which our course does not.

The specific use of images has been largely limited to developing skills in piece-wise programming concepts like file I/O [7,16], arrays, functions, etc. [3,4,7]. Images have also been used to illustrate advanced data structures and their use in practical algorithms [9], and also in program design and testing [10]. While these are valid and interesting examples in programming, our work uses images as an underlying and continuing theme in a programming course. We attempt to teach students both programming and elements of high-level application design while still developing programs from scratch. In order to concentrate on programming rather than domain knowledge we provide students with only an implementation-specific view of these algorithms. Although such a treatment may not be appropriate for computer science students or in advanced programming courses [17] we feel it is ideal for an audience of information systems students as it provides high yield (i.e. working programs with images) without excessive technical knowledge.

3. Details of our experiment

To conduct this experiment we chose a course (IT 275: Java as a Second Language) that is designed primarily to provide experience in Java to students transferring from other colleges and graduate students lacking programming experience. Students taking this course are required to have at least a semester's worth of experience in a high-level programming language (not necessarily object-oriented). This course typically comprises of majors from information systems with possibly a few computer science students transferring from other smaller colleges. This course (worth 4 credit-hours) offers an alternative (mostly for transfer and graduate students) to taking two courses (3 credit-hours each) that introduce Java and object-oriented concepts to undergraduate IS majors starting at our university. An overview of the course is provided in Table 1.

Table 1
Overview of our course.

Week	Topics
1,2	Introduction, classes, objects
3	Conditionals and loops
4,5,6	Arrays, multidimensional arrays, method overloading
7–8	Inheritance and polymorphism in Java
9	Exception handling
10,11	GUI in Java Swing
12–13	Generics and collection classes
14	Multithreading
15	Java and XML, Javadocs and jar files

Author's personal copy

From earlier experience in teaching this course we observe that students struggle with the significant breadth of topics that it covers. We attribute this to two factors: its inherent role as a "make-up" course before taking other courses directly related to the degree programme and a biased view that programming is difficult and not critical to being a good information analyst. We use image processing to achieve the twin objectives of covering the significant breadth of topics in a cohesive manner and to kindle students' interest in programming for practical problems.

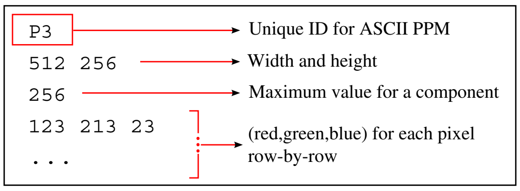
Initially we did not disclose the fact that students were to work with images. This was done so that students do not feel apprehensive after hearing about technical jargon and the underlying mathematics. Not all assignments in this course involved images. We felt this was important so that if some students do not find images interesting as we hoped, the course would not seem completely inaccessible to them. Since each assignment (and its code) built upon previous assignments we were concerned that students would gradually fall behind. We mitigated this in two ways: (1) including copious comments with the grade of each assignment detailing their specific errors so that they do not waste time debugging previous errors, (2) having 1:1 discussions with students and helping them with their code, mostly during regularly scheduled office hours. We deliberately decided not to provide ready-made solutions to our assignments because we wanted students to approach us and work through their problems instead of ignoring their errors and moving on. After each assignment was submitted, we had a discussion with students in class about the details of the algorithm that they implemented and its practical use. We also asked for their informal feedback.

3.1. *Assignment 1: Working with PPM images*

Learning objective (s):	Work with text file I/O, create and manipulate multi-dimensional arrays
Duration:	1 week

Students were asked to create a simple class that stored integer data in a 3D array. They were provided with a narrative explanation of the Portable Pixmap (.ppm) file format [25], along with several example files in that format (Fig. 1 shows an example). They were asked to write methods to read a file (*fromFile*) into a 3D array, flip the 3D array across the first dimension (*flip*, flip the image vertically) and write the 3D array to a .ppm file in the correct format (*toFile*).

Experience: The PPM file format was chosen because it is extremely simple and ASCII-based, and thus helpful in debugging. Many standard I-P programs like Photoshop [18] and GIMP [19] support it. This assignment was given to students in a lab environment to acclimatize them with the programming environment. A pre-compiled program was provided that checked whether the files they produced were valid flips of each other. After the lab was over, the results produced by their programs were revealed to be images and were visualized in Photoshop. As many students initially have difficulty understanding applications of arrays with more than two dimensions, the realization that color images can be represented simply as 3D arrays was interesting for many students.



```
P3
512 256
256
123 213 23
. . .
```

Unique ID for ASCII PPM
Width and height
Maximum value for a component
(red,green,blue) for each pixel row-by-row

Fig. 1. The ASCII-based PPM file format.

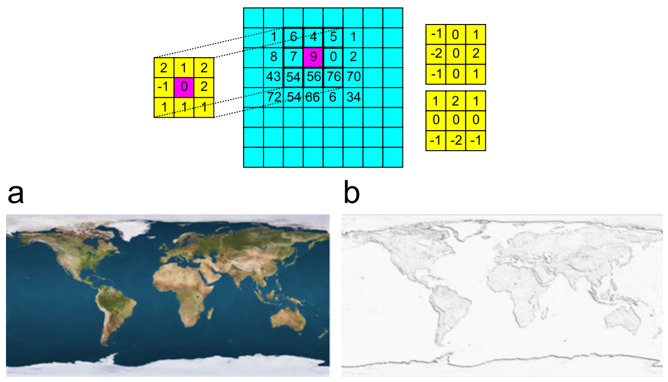


Fig. 2. Image filters: top row: convolution with a 3 × 3 filter. The two filters on the right represent the Sobel edge detector. Bottom row: (a) original image and (b) edge-detected result (inverted for illustration).

3.2. *Assignment 2: Image filters*

Learning objective(s):	Write loops correctly, manipulate arrays
Duration:	1 week

This assignment was named "Operations on Arrays", and asked students to write several methods in the class that they wrote earlier. Specifically they were to write a method (*filter*) to implement a 3 × 3 filter by placing the center of the filter at a provided location in the image, and convolving it with the image (see Fig. 2) (considering only those parts of the filter that overlapped an image pixel, to address the issue of pixels at the edges of the image). They were to write another method (*getEdges*) that applied this filter at every location in the image, for each color channel (the third dimension of the array). They would have a third method (*thresholdForDisplay*) to clamp all the numbers to the range 0–255 so that Photoshop can display it. Finally students were provided with 3 × 3 Sobel filters [20] that produced an edge-detected image as the result (shown in Fig. 2, bottom row).

Experience: As Fig. 2 illustrates, the convolution filtering algorithm can be expressed succinctly using pictures. After a few attempts, many students figured out what the expected output would be and started using Photoshop as a verification tool. Some were familiar with similar operations that Photoshop had to offer.

3.3. *Assignment 3: Compressing artifacts using wavelet transforms*

Learning objective(s):	Write more complicated loops, work on sub-parts of arrays
Duration:	2 weeks

The main objective of this assignment was to provide practice in writing more complicated loops. This was motivated by our observation that although students were well-versed in the syntactic details of loops they often could not correctly use them to solve a given problem. This assignment also asked students to work on sub-parts of arrays by choosing and manipulating appropriate indices.

Students used 2D Haar wavelets (see Fig. 3(b–e)) to perform multi-resolution analysis of images, with the purpose of simulating compression artifacts. Wavelets break a signal into base (or coarse approximation) and detail (difference between base and

Author's personal copy

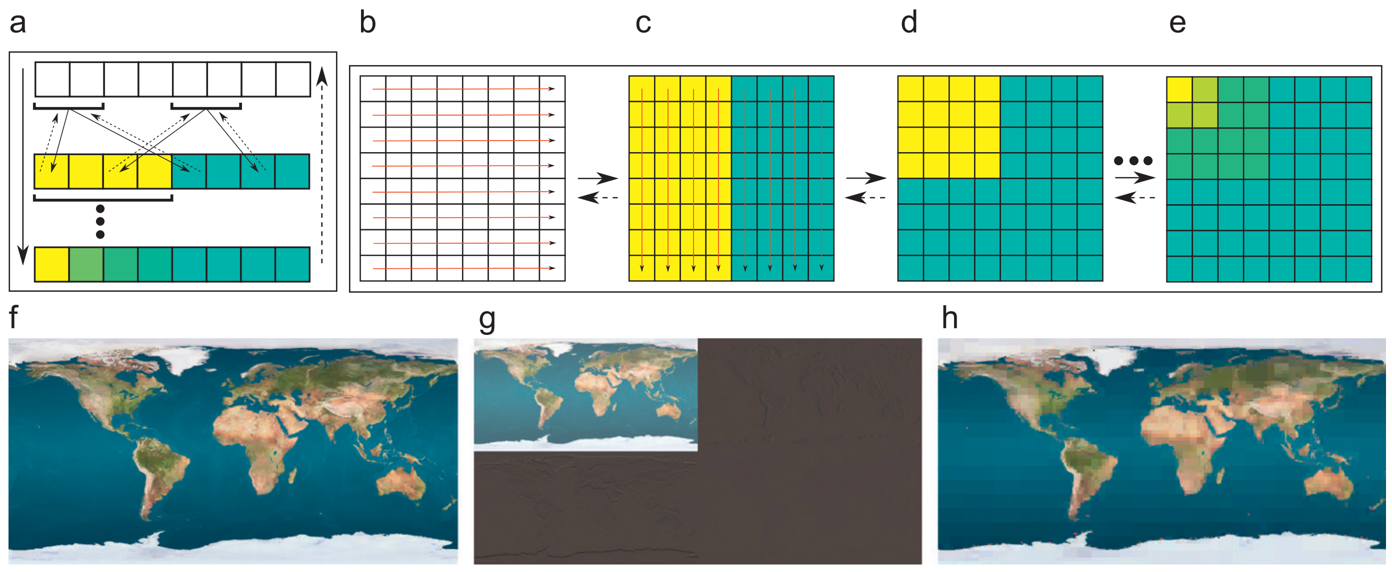


Fig. 3. Multiresolution analysis. (a) 1D Haar wavelets. In the first iteration, successive pairs of values in an array are used to populate another array that contains base (yellow) and detail (cyan) values. This process is recursively applied to the base-part until it reduces to size 1 (bottom). (b–e) (Non-standard) 2D Haar wavelets. In the first iteration the 1D transform is applied to all rows (b,c) followed by all columns ((c)→(d)) resulting in base (yellow) and detail (cyan) values. This process is recursively applied to the base-part until it reduces to size 1×1 (e). (f–h) Illustration on an example image of size 1024×512. (f) The original image. (g) Illustration of the result after one iteration (i.e. corresponding to step (d)). (h) Compression artifacts created by transforming, reducing all details below 0.4–0 and inverting the transform. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

actual signal) coefficients. If some or all the details are lost, an inverse transformation does not yield the original image, but one with visible compression artifacts (Fig. 3(h)).

A 1D Haar wavelet transform is applied to a 1D array whose dimension is a power of 2 as follows:

1. Read the array two numbers at a time, say p and q.
2. Compute two numbers $b=(p+q)/\sqrt{2}$ and $d=(p-q)/\sqrt{2}$.
3. Create a new array whose first half consists of all b's (base) and the other of all d's (details).
4. Recursively apply above steps to the "base" sub-array.

An inverse Haar wavelet can be applied by starting from the array obtained in step 3, reading two corresponding numbers from the two halves of the array, applying the same equations in step 2 to them, and storing them at successive positions in the new array. Fig. 3(a) illustrates the process. A "non-standard" 2D Haar wavelet is simply the application of steps 1–3 above to every row, followed by every column of the 2D array.[1] We refer the interested reader to Gonzalez et al. [20] for an in-depth overview of the Haar wavelet transform.

Students had to apply the 2D Haar wavelet transform to an image, threshold the detail coefficients and invert the transformation to see the results. We provided actual examples of the 1D array transform to help them to test their programs.

Experience: This was designed to be one of the more difficult assignments of the semester, and therefore students asked for and were provided more hints and help in this assignment than any other. We felt that a step-by-step illustration of the algorithm (i.e. a 1D Haar wavelet transform, followed by a 2D transform and inverse transforms) helped them to implement it correctly. Once again students were told about the expected outcome only in words (blocky artifacts in images). As a hint, they were encouraged to use the *thresholdForDisplay* method developed in Assignment 2 to visualize individual iterations of their transforms (e.g. visualize Fig. 3(g)). Many students found such "visual" debugging useful, while some struggled with it. This was addressed during 1:1 discussions by showing them examples of how their code affected the image visually.

3.4. Assignment 4: Basic program design with menus and the undo-redo feature

Learning objective(s):	Introduce MVC architecture, use inheritance and polymorphism for good design
Duration:	2 weeks

This assignment changed the focus from technical operations to overall program design. The first objective of the assignment was to introduce the model-view-controller (MVC) architecture. Students had to write a handler class that acted as the "controller" by working with both the user interface (a text-based user interface for this assignment, as shown in Fig. 4) and the actual *Image* class. The second objective was to provide them with an example of using inheritance and polymorphism. This was done by asking students to design all the image operations in a streamlined manner using the *command* design pattern [21]. This allowed them to regard all these operations in a general manner. This point of view motivated the undo–redo mechanism in terms of general operations. The third objective was to teach them to think how a particular data structure (in this case, a stack) is suited for specific functionality (in this case, the ability to undo and redo operations).

This was the only assignment in which they were provided with some existing code. In order to compensate for students' lack of knowledge of data structures a simple stack implementation was provided to them, along with a short explanation. Students were expected to observe how a stack worked and relate its behavior to the intended behavior of the "undo" and "redo" operations. We asked students to implement the undo–redo feature by simply writing the "before" and "after" versions of an image for every operation to files.

Experience: This assignment generated an interesting discussion in class about how to undo certain operations. While the *flip* operation can be undone by flipping once again, the

[1] A standard Haar wavelet transform completes the row iterations before the column iterations. A non-standard transform visualizes better.

Author's personal copy

edge-detection operation cannot. This showed students how different mechanisms may be necessary within the same program to implement one feature (e.g. the undo mechanism). Some students implemented the undo mechanism by maintaining a stack of *Image* objects directly. While this was conceptually correct, they experienced an ‘‘Out of memory’’ error after they specified 4–5 operations in succession without undoing any of them. This was attributed to the large size of some images resulting in large objects being pushed on the stack. This generated an even more interesting discussion since most students were unfamiliar with memory management issues, thanks to Java's automatic garbage collection, and hence had never encountered such an error before. Upon subsequent investigation we found documentation on how GIMP [22] defines separate mechanisms for undoing operations, and allows the user to customize its undo–redo capability.

3.5. *Assignment 5: Graphical user interface*

Learning objective(s):	Design GUIs using Swing without WYSIWYG utilities, exploring Java documentation
Duration:	2 weeks (2 more weeks for extra credit)

This assignment was designed to give students practice in developing user interfaces in Java. Students were not allowed to use WYSIWYG tools (e.g. Netbeans [23] for Java) for this purpose–they were expected to write all the code themselves and use existing layout managers. A secondary objective was to make them rely more on documentation to determine which classes to use and how. This was an important goal towards the end of such a programming course so that students become more self-reliant

```
Welcome to my small image manipulation program.
Main menu:
'o'          :Open a PPM image.
's'          :Save the current image in PPM format.
'p'          :Print name of the current image.
'f'          :Flip the current image vertically.
'e'          :Find edges in current image.
'c'          :See compression artifacts in image.
'u'          :Undo.
'r'          :Redo.
'x'          :Exit the program.
Please select an option:
```

Fig. 4. Assignment 4: text interface for program. A user would save a file after processing it and then use Photoshop to view it.

rather than expecting to be taught everything by instructors. At the end of this assignment, students had created a fully functional image manipulation application with three image operations and the undo–redo feature.

Students were asked to replace the text-based interface made in Assignment 4 (Fig. 4) with a GUI that matched the layout provided to them as a screen capture (Fig. 5). We provided hints about specific Java classes that would be useful in this assignment. For extra credit they were asked to use the Java Swing API to read and convert between standard image formats (i.e. *.bmp*, *.jpg*, etc.) and their own Image class. This assignment generated enthusiasm among students as it enabled them to finally see and work with images in their own program.

4. Evaluation

4.1. *Comparative quantitative evaluation*

In order to evaluate the effect of our I-P theme on student performance we compared the average performance of students in three sections of the same course taught in different semesters. The first and second sections (referred henceforth as S_{r1} and S_{r2} respectively) had a total of five unrelated assignments and a project. The third section (referred henceforth as S_{image}) had a total of seven assignments (five of which were I-P related) and two lab sessions (many of these assignments were smaller than those in the other two sections). S_{r1} and S_{image} had two exams and were taught by the author, while S_{r2} had three exams and was taught by a different instructor. All sections comprised of a mixture of undergraduate and graduate students, and were mostly information systems majors (with some computer science majors).

Fig. 6(a) shows the comparison between percentage average performance of students across five unrelated assignments in S_{r1} (blue), S_{r2} (green) and the 5 I-P assignments in S_{image} (red). Each corresponding assignment in the three sections had roughly similar objectives, although they were unevenly distributed because S_{image} had more but smaller assignments. This comparison shows that although S_{r2} is different, S_{r1} and S_{image} followed roughly similar trends in performances from one assignment to the next. It also shows that, overall, students in S_{image} with the image processing assignments performed slightly better than those in S_{r1} and S_{r2}. Fig. 6(b) compares the performance trend of students in S_{r1} (blue), S_{r2} (green) and S_{image} (red) across all assignments and exams spanning the semester (*X*-axis). The black dots denote the averages of exams during the semester, while the green dots reflects those of the final exams in

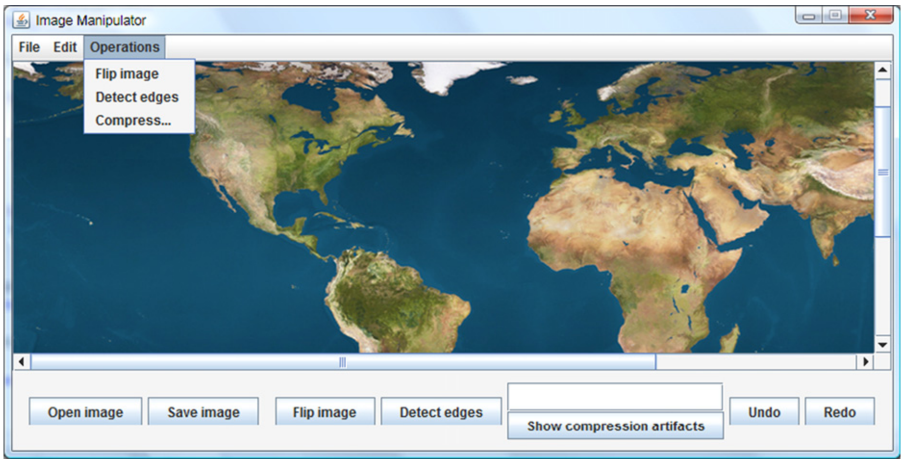


Fig. 5. Students replicated this GUI using Java Swing.

Author's personal copy

the three sections. Fig. 6(c) shows a summary of letter grades (expressed as a percentage of the number of students). All sections were graded on an absolute basis. It shows that although more students in S_{image} received worse grades, it also got more A's. The absence of grades lower than 'B' in S_{r1} can be attributed to two reasons. First the minimum cutoff for a 'B' was lower by one percentage point in S_{r1} than in S_{r2} and S_{image}. Secondly, S_{r1} had approximately 70% graduate students whereas S_{image} had about 50%, which likely increased overall class performance in the former.

Although the above comparison shows only a marginal increase in performance, it does not take into consideration other benefits of our approach, namely progressively creating a single application from scratch and exposing students to practical image processing algorithms from an implementation point of view. Although it is difficult to measure the influence of these factors to student learning, we believe that these aspects contribute positively to what our students could implement and accomplish in a single semester. As it is included in an introductory course such material also helps students to broaden their awareness and understanding of other subjects within information technology.

We also attempted to assess whether students felt that they accrued the overall benefits of this course even when working with specific I-P related material. Every course in our school undergoes a standardized IDEA assessment [24]. Table 2 reports and compares some details from student feedback for S_{r1} and S_{image}, respectively. The objectives were collectively set by the faculty who teach this course. Although factors like student background and teaching styles also affect such responses (e.g. S_{r1} and S_{image} represent the first and second time respectively that the author taught this course), these results nevertheless show a greater percentage of students concurring that course objectives were met when I-P assignments were introduced.

4.2. Qualitative evaluation

In general students were enthused with the idea of working with images. This was especially evident after the first assignment when they realized that the text files provided to them were images. After the first assignment, as soon as an assignment was posted a few students regularly searched online for the details of the algorithm that they were asked to implement and contacted us to confirm what they "guessed" they were implementing. This showed us that students were motivated to complete the assignments. We received quite a few questions about the details of the underlying algorithms: "*how does the edge detection actually work*?", "*is this what Photoshop uses* (*for edge detection*)?", "*what do the numbers in the filters actually mean*?", "*are wavelets practically used for image compression*, *and if so*, *where*?". Soon after assignment 3 was posted one student contacted the instructor saying he searched online, found "*something called wavelets*" and "*thought it looked really complicated*". That seemed to support our theory that hiding mathematical details indirectly helps students to complete assignments with less apprehension. Assignment 4 generated useful discussion about design. Some students wondered if there was a better way to implement the mechanism without resorting to files that took up unnecessary space on the hard drive. One student implemented an undo–redo mechanism that preserved the order of operations across several images (i.e. his program regarded the file "open" and "close" operations as undoable as well, making it possible to revert back to an earlier opened image and undoing its operations. This is not possible in most commercial image manipulation programs.). Some students described the overall experience of the course as ". .*great way to improve critical thinking with just about sufficient help from the instructor*", "*good job fitting students from different backgrounds*", "*great class*", "*images were useful and more importantly we learned Java while working on them*", etc. There were a few comments about how the nature of the assignments made the course difficult for them because the basic idea of working with images did not excite them. A few students struggled with manipulation of 3D arrays as they had never encountered them.

In future, we plan to evaluate our experiment more formally by including student surveys at regular intervals in the course. This would give us more progressive and precise feedback regarding impact on student learning. We also plan to solicit the feedback of

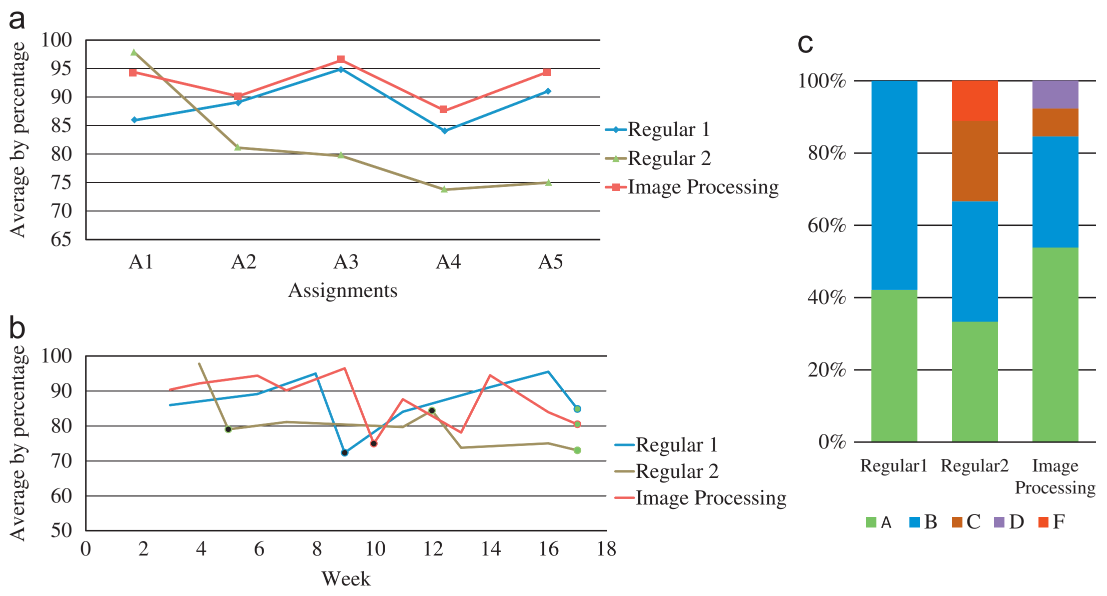


Fig. 6. Performance trends comparing three course sections in different semesters without (blue and green in (a–b)) and with (red in (a–b)) I-P assignments respectively. (a) Average performance on assignments (each assignment is worth 100 points). The courses shown in blue and green had a larger project as the last assignment. Although the same approximate trends were followed, the overall performance was positively affected because of the I-P assignments. (b) Average performance trend over the entire semesters for the three evaluated sections. The black dots represent the mid-semester exam(s), while the green dots represent the final exams of both sections. (c) Plot of letter grades awarded for both sections as percentages of their respective sizes. The semester with I-P assignments had a greater number of A grades, although the lowest grades were worse than those in one of the semesters with no such assignments. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Author's personal copy

Table 2
Results from IDEA evaluations. The rating of each objective were given by faculty who regularly teach this course. The responses show that most students perceive that the objectives considered most important were fulfilled in this experimental way of teaching it, and that this perception was greater compared to when I-P assignments were not used. (Note: Responses to each question in a section do not add up to 100% because (i) IDEA does not report statistics about the rating of 3 (ii) not all surveyed students answered every question.).

Question	Rating	Percentage of students rating (out of 5)			
		Without I-P		With I-P	
		1–2	4–5	1–2	4–5
Gaining factual knowledge (terminology, classifications, methods, trends)	Important	13	60	0	90
Learning fundamental principles, generalizations, or theories	Important	0	50	0	90
Learning to *apply* material (to improve thinking, problem solving, and decisions)	**Essential**	27	53	0	80
Developing specific skills, competencies, and points of view needed by professionals in the field most closely related to this course	**Essential**	20	53	0	78

instructors teaching subsequent courses to determine if students have demonstrated the expected programming skills.

5. Conclusions

We had two main objectives in mind when teaching this course using the I-P theme. The first objective was to make programming more interesting to students who do not view it so, assuming that greater student interest would lead to better performance in class and deeper understanding and appreciation of the subject. Introductory programming courses often teach such basic concepts that it is difficult to design assignments that are simultaneously simple, challenging and interesting. We feel an image processing theme provides all three aspects. Our interactions in the classroom, along with the average performance of the students in the assignments, provide encouraging evidence that our approach worked well in these aspects. Creating the application progressively also allowed greater modularity and flexibility in how the assignments were set up.

The second objective was to use image processing purely as a pedagogical tool without deviating from the more general goals of the course. This was especially important since our students are not likely to encounter I-P due to their choice of major. Our I-P assignments directly mapped to many of the topics explicitly covered in this course. Some unrelated assignments, originally included to prevent focusing too much on image processing, also helped in covering course topics that did not map well to the I-P assignments. We highly recommend this hybrid approach when adopting our theme in other programming courses.

Many variations on the above assignments can be designed to expose students to other image effects. Convolution filtering can be used for effects such as blurring and affine transformations on color for sepia toning. Similar to the wavelet transform, other seemingly "complicated" algorithms like the Fourier transform can be used to create interesting effects. More design patterns, such as *proxy* and *adapter*, can be introduced to emphasize more on design aspects.

Adopting the image processing theme was easier and more intuitive as we had a background in computer graphics and visual computing. However, when we encouraged other instructors to adopt our approach, they were concerned about whether their insufficient image processing knowledge would have a negative effect on their teaching. In our experience good textbooks on image processing (e.g. [20]) explain operations in algorithmic form with enough detail to directly translate into tractable programming assignments. Verification of correctness is both visual and simple in many cases. Thus similar to students, instructors can also view I-P operations purely from an implementation perspective without being experts in the field.

Ongoing and future work: After this experience with IT 275, a section of a traditional IS programming course mandatory for all IS undergraduate students was taught using an image processing theme. Some of the variations discussed above were given as assignments (e.g. blurring and sepia toning). However the size of the section was too small to perform a reliable quantitative analysis of student performance. We wish to continue performing this experiment in such traditional courses to evaluate whether such an approach is viable for a course that is mandatory for all students. Although only one section of IT 275 is offered in a semester, it would be worthwhile performing this experiment on some but not all sections of the same course during the same semester. This could help us to better understand its impact on students' learning in a comparative way.

Acknowledgments

We thank Dr. Yongning Tang for providing data for one of the sections of this course for comparative evaluation. We also thank our students for motivating us to try interesting approaches to teaching programming and responding enthusiastically to them.

References

[1] Google picasa. 2011. ⟨http://picasa.google.com⟩.
[2] Flickr. 2011. ⟨http://www.flickr.com⟩.
[3] Astrachan O, Rodger SH. Animation, visualization, and interaction in cs 1 assignments. In: Proceedings of SIGCSE; 1998. p. 317–21.
[4] Burger KR. Teaching two-dimensional array concepts in java with image processing examples. SIGCSE Bull 2003;35(1):205–9.
[5] Davis TA, Geist R, Matzko S, Westall J. $\tau\epsilon\chi\nu\eta$: a first step. In: SIGCSE; 2004. p. 125–9.
[6] Davis T. Teaching data structures and algorithms through graphics. In: Proceedings of Eurographics—education papers; 2007. p. 33–40.
[7] Fell H, Proulx V. Exploring martian planetary images: C++ exercises for cs1. SIGCSE Bull 1997;29(1):30–4.
[8] Hunt K. Using image processing to teach cs1 and cs2. SIGCSE Bull 2003;35(4):86–9.
[9] Jiménez-Peris R, Khuri S, Pati No-Martínez M. Adding breadth to cs1 and cs2 courses through visual and interactive programming projects. In: Proceedings of SIGCSE; 1999. p. 252–6.
[10] Wicentowski R, Newhall T. Using image processing projects to teach cs1 topics. SIGCSE Bull 2005;37(1):287–91.
[11] Shesh A. High-level application development for non-computer science majors using image processing. In: Proceedings of Eurographics—education papers; 2011. p. 29–36.
[12] Matzko S, Davis T. Using graphics research to teach freshman computer science. In: SIGGRAPH education program; 2006.
[13] Leutenegger S, Edgington J. A games first approach to teaching introductory programming. In: Proceedings of SIGCSE; 2007. p. 115–8.
[14] Jordi L, Esparaza J. Computer graphics for information system programmers. In: Proceedings of Eurographics—education papers; 2010. p. 57–62.
[15] Guzdial M. A media computation course for non-majors. SIGCSE Bull 2003;35:104–8.
[16] Urness T. Teaching file input/output, loops, and if-statements via a red eye reduction assignment. J Comput Small Colleges 2008;23(4):286–90.
[17] Sutherland KT. Image processing programming projects in an upper division algorithms course abstract. In: Midwest instruction and computing symposium; 2004.
[18] Adobe photoshop ⟨http://www.photoshop.com⟩, 2011.

Author's personal copy

[19] Gimp 2011. ⟨http://www.gimp.org⟩.
[20] Gonzalez R, Woods R. Digital image processing.Prentice-Hall; 2008.
[21] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns.Addison Wesley; 1995.
[22] Undoing in Gimp. 2011 ⟨http://docs.gimp.org/en/gimp-concepts-undo.html⟩.
[23] The Netbeans IDE. 2011 ⟨http://netbeans.org⟩.
[24] The IDEA center. 2011 ⟨http://www.theideacenter.org/⟩.
[25] PPM format. 2011 ⟨http://netpbm.sourceforge.net/doc/ppm.html⟩.