High-Level Application Development for non-Computer Science majors using Image Processing

Amit Shesh†

Illinois State University

Abstract

In many ways it is a unique challenge to teach programming and high-level application development to non-computer science majors like information systems. Simple visual computing can be a very helpful tool in such situations because it enables programs to produce something students can see. 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 implementing 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. This allowed us to satisfy all the objectives of a typical programming course while simultaneously exposing students to developing meaningful applications from scratch with "standard" features. Our classroom was comprised of a mix of undergraduate and graduate students lacking sufficient programming background. With minor variations, our approach can be fit to courses for other majors where programming is considered useful but not critical.

Categories and Subject Descriptors (according to ACM CCS): K.3 [Computers and Education]: Computer and Info. Science Education—I.4 [Image Processing and Computer Vision]: —D.2.11 [Software Architectures]: Patterns—

1. Introduction

In contrast with computer science for which programming is a critical skill, many IT-related majors such as information systems (IS) place at best a secondary importance to programming. Many students of such majors mistakenly view programming courses as something they must "survive". At our university we see many students choose a non-computer-science program simply due to fear of programming. In the classroom this poses a challenge for instructors who must motivate such students with innovative pedagogical techniques. Moreover an increasing number of universities offer courses that are cross-listed across different majors. This increases the diversity in technical backgrounds that students bring to a typical programming course which compounds the challenge of making course material interesting, comprehensible and "immediately applicable" to all.

Visual computing 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. This can be addressed by providing supporting code that simplifies and hides technical details, but this may distract a course from its original objectives. Moreover we contend that writing applications from scratch provides students with a comprehensive understanding of how they work. Therefore we work with static images and design projects that provide a balance between simplicity and practicality. Images provide natural examples of arrays and their manipulation and thus provide enough technical challenge.

Although images have been popular teaching tools in computer science [AR98, Bur03, DGMW04, DD07, FP97, Hun03, JPKP99, WN05], 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 ex-

† ashesh@ilstu.edu

ample, 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 implementation-specific issues?

Image processing, like most forms of visual computing is very mathematical. Many non-CS majors do not include or emphasize less on data structures, algorithms and mathematics. Introducing math 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 experiment shows some promise in this direction, as students were motivated to complete their assignments in many cases with little idea about why 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 such aspects. We used private and classroom discussions to encourage those students who were more interested in learning the underlying algorithms.

2. Related Work

Visual computing in the form of computer graphics [DGMW04] or image processing [MD06] has been used in programming courses, but primarily in computer science where the expected knowledge of math is higher. Leutenegger *et al.* [LE07] use games as tools to teach programming for computer science students, but using multimedia-focused languages. Jordi *et al.* [JE10] discuss the use of computer graphics to teach information systems students, the same audience as ours. Guzdial [Guz03] designed a course for non-CS majors that focuses on multimedia computation. Although the foundation of much such work in this area, their course uses significant existing material for students to use and extend which our course does not.

Images have been used to develop skills in piece-wise programming concepts like file I/O [FP97, Urn08], arrays, functions, etc. [AR98, Bur03, FP97], as well as algorithm and program design and testing [WN05]. 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.

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. The 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.

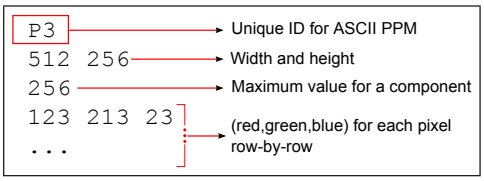


Figure 1: *The ASCII-based PPM file format.*

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 program 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.

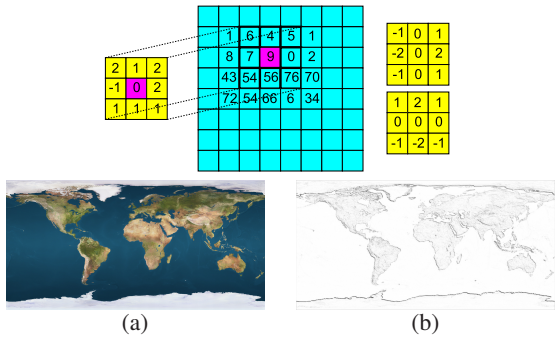
Initially we did not disclose the fact that students were to work with images. This was done so that students do not feel apprehensive after hearing about technical jargon and the underlying mathematics. After each assignment was submitted, we had a discussion with students in class about the details 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, declare and use multi-dimensional arrays
Duration	:1 week

Students were asked to create a simple class that stored integer data in a 3-dimensional array. They were provided several files in the Portable Pixmap (.ppm) file format (Figure 1 shows an example). They were supposed to write methods to read a file(*readPPM*), flip the 3-dimensional array across the first dimension(*flip*, flip the image vertically) and write the 3D array to a .ppm file in the correct format(*writePPM*).

Experience: The PPM file format was chosen because it is ASCII-based and thus is helpful in debugging. Many standard image processing programs like Photoshop and GIMP [gim] 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 images 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.



(a) (b)

Figure 2: *Image Filters: Top row: Convolution with a* 3×3 *filter. The two filters on the right represent the Sobel edge detector. (a) original image (b) edge-detected result (inverted for illustration).*

3.2. Assignment 2: Image Filters

Learning Objective(s)	:Write loops for multi-dimensional arrays
Duration	:1 week

Filtering by Convolution: 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 Figure 2). They were to write another method (*convolve*) that applied this filter at every location in the image, for each color channel (the 3^{rd} 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 [GW08] that produced an edge-detected image as its result (shown in Figure 2, bottom row).

Experience: 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)	:Use more loops with arrays, working with sub-parts of arrays
Duration	:2 weeks

The main goal 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. Students used 2D Haar wavelets(see Figure 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 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 (Figure 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 2 numbers at a time, say p and q.
2. Compute two numbers $b = \frac{p+q}{\sqrt{2}}$ and $d = \frac{p-q}{\sqrt{2}}$.
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 2 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. Figure 3(a) illustrates the process. A 2D Haar wavelet is simply the application of steps $1-3$ above to every row, followed by every column of the 2D array. We refer the interested reader to Gonzalez *et al.* [GW08] 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 transformations to see the results. We provided actual examples of the 1D array transform to help them to test their programs.

Experience: Once again students were told about the expected outcome only in words ("blocky artifacts in images"). Some students used the *thresholdForDisplay* method developed in Assignment 2 to visualize individual iterations of their transforms (e.g. visualize Figure 3(g)). Many students found such "visual" debugging useful, while there were some who struggled with it.

3.4. Assignment 4: Basic Program Design with Menus and the Undo-Redo Feature

Learning Objective(s)	:Using MVC architecture, use inheritance and polymorphism in good design
Duration	:2 weeks

With this assignment the focus was changed 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 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 [GHJV95]. 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

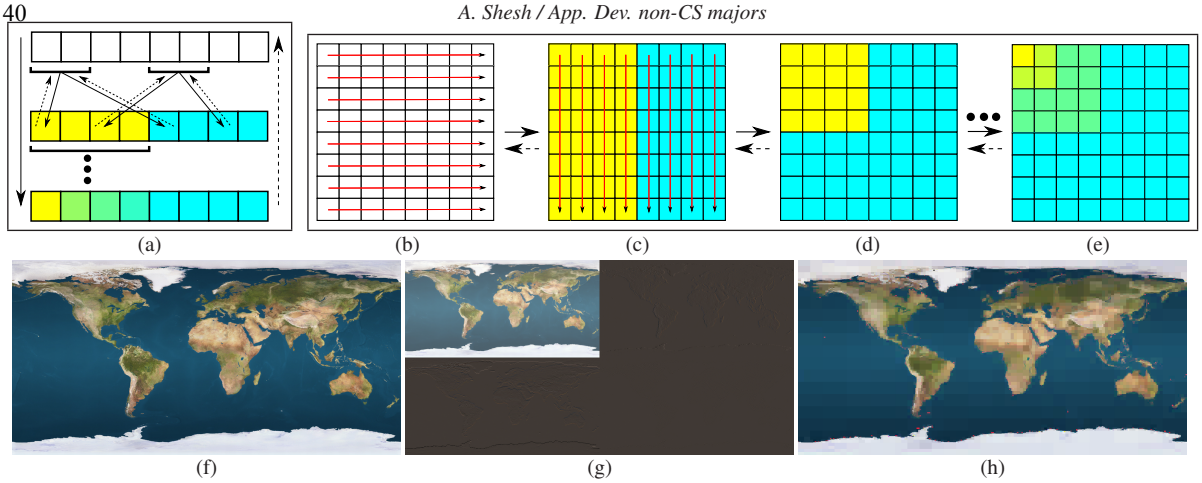


Figure 3: *Multiresolution analysis. (a) 1-D haar wavelets. In the first iteration, successive pairs of values in an array are used to populate another array that contains base (yellow) and detail (cyan) values. This process is recursively applied to the base-part until it reduces to size 1 (bottom). (b-e) 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 *to* 0 *and inverting the transform.*

explanation. 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 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 the open-source GIMP program that actually defines separate mechanisms for undoing several operations, and how it 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 for this purpose–they were expected to write all the code themselves and use existing layout managers. A secondary objective was to make them

```
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:
```

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

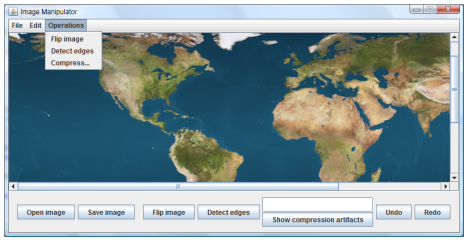


Figure 5: *Students replicated this GUI using Java Swing.*

rely more on documentation to determine which classes to use and to solve a given problem. This was an important goal towards the end of such a programming course so that students become more self-reliant 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.

Students were provided with a screen shot of the desired layout (Figure 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. Overall Feedback

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 lesser apprehension. Assignment 4 generated useful discussion about design. Some students wondered if there was 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 undo-able 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 focussed and precise feedback on whether this theme has an impact on student learning. We also plan to solicit feedback of instructors teaching subsequent courses to determine if students have demonstrated the expected programming skills.

5. Variants and Future Work

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 (i.e. old photographs that appear "brown-toned"). Similar to the wavelet transform, other seemingly "complicated" algorithms like the Fourier transform can be used to create interesting effects. More design patterns can be introduced to emphasize more on design aspects, such as *proxy* and *adapter*.

In the future we wish to perform this experiment in more traditional IS programming courses. Performing this experiment on some but not all sections of the same course could help us to better understand its impact on students' learning in a comparative way.

References

[AR98] ASTRACHAN O., RODGER S. H.: Animation, visualization, and interaction in cs 1 assignments. In *Proc. SIGCSE* (1998), pp. 317–321.

[Bur03] BURGER K. R.: Teaching two-dimensional array concepts in java with image processing examples. *SIGCSE Bull. 35*, 1 (2003), 205–209.

[DD07] DUCHOWSKI A., DAVIS T.: Teaching algorithms and data structures through graphics. In *EG Edu. Papers Program* (2007).

[DGMW04] DAVIS T. A., GEIST R., MATZKO S., WESTALL J.: τεχνη: a first step. In *SIGCSE* (2004), pp. 125–129.

[FP97] FELL H., PROULX V.: Exploring martian planetary images: C++ exercises for cs1. *SIGCSE Bull. 29*, 1 (1997), 30–34.

[GHJV95] GAMMA E., HELM R., JOHNSON R., VLISSIDES J.: *Design Patterns*. Addison Wesley, 1995.

[gim] GIMP. http://www.gimp.org.

[Guz03] GUZDIAL M.: A media computation course for non-majors. *SIGCSE Bull. 35* (2003), 104–108.

[GW08] GONZALEZ R., WOODS R.: *Digital Image Processing*. Prentice Hall, 2008.

[Hun03] HUNT K.: Using image processing to teach cs1 and cs2. *SIGCSE Bull. 35*, 4 (2003), 86–89.

[JE10] JORDI L., ESPARAZA J.: Computer graphics for information system programmers. In *EG Edu. Papers Program* (2010).

[JPKP99] JIMÉNEZ-PERIS R., KHURI S., PATI NO-MARTÍNEZ M.: Adding breadth to cs1 and cs2 courses through visual and interactive programming projects. In *Proc. SIGCSE* (1999), pp. 252–256.

[LE07] LEUTENEGGER S., EDGINGTON J.: A games first approach to teaching introductory programming. In *Proc. SIGCSE* (2007), pp. 115–118.

[MD06] MATZKO S., DAVIS T.: Using graphics research to teach freshman computer science. In *SIGGRAPH 2006 Edu. Program* (2006).

[Urn08] URNESS T.: Teaching file input/output, loops, and if-statements via a red eye reduction assignment. *J. Comput. Small Coll. 23*, 4 (2008), 286–290.

[WN05] WICENTOWSKI R., NEWHALL T.: Using image processing projects to teach cs1 topics. *SIGCSE Bull. 37*, 1 (2005), 287–291.