# Introductory Programming using Processing

Mark M. Meysenburg
Information Science and Technology Department
Doane College
Crete, NE 68333
mark.meysenburg@doane.edu

## Abstract

For the past two years, we have used the Processing programming language to teach our introductory programming (CS1) course at Doane College. Processing [10] is a freely-available programming language derived from Java. The language is designed to allow programmers to easily add graphics, animation, sound, and user interaction to their programs. These characteristics make the language an excellent choice to teach during a first programming course, particularly at a small liberal arts college. This paper describes the philosophy of our course, details why we are using Processing, illustrates several concrete examples of assignments and projects we have used, and presents, subjectively, the impact teaching with Processing has had on our students.

# 1 Introduction

For the past two years, we have used the Processing programming language to teach our introductory programming (CS1) course at Doane College. Processing [10] is a freely-available programming language derived from Java. The language is designed to allow programmers to easily add graphics, animation, sound, and user interaction to their programs. These characteristics make the language an excellent choice to teach during a first programming course, particularly at a small liberal arts college. This paper describes the philosophy of our course, details why we are using Processing, illustrates several concrete examples of assignments and projects we have used, and presents, subjectively, the impact teaching with Processing has had on our students.

# 2 The Doane College CS1 Course

Doane College is a small liberal arts college in Crete, Nebraska. Our Information Science and Technology department offers a Computer Science major, an Information Science major, a Computer Studies minor, and an Information Technology Teaching Endorsement. Each of these starts with a four-course core sequence:

- IST 140 Introduction to Information Science and Technology,
- IST 145 Introduction to Programming and Problem Solving,
- IST 146 Programming and Problem Solving II, and
- IST 252 Principles of Digital Logic and Computer Organization.

IST 145 is our "CS1" course, where we introduce students to the fundamentals of computer programming. Due to faculty constraints and student demand, only one section of IST 145 is offered each semester, and IST 145 is the only introductory programming course taught at Doane. Further, IST 145 is a service course for the Mathematics and Physics departments; their students must take IST 145 as a cognate for their majors. Since Doane is a liberal arts college, many of our CS / IS majors are also involved in humanities pursuits, such as art, music, or theater; computing is usually not their only passion. The internal and external constraints on our CS1 course, plus the characteristics of our liberal arts students, have influenced our decision to adopt Processing as the language for CS1, and also the method in which we teach programming in the course.

To serve the needs of our CS / IS students, as well as the needs of students taking IST 145 as a cognate, we teach our course in a modified "objects later" manner, instead of "objects first." For our purposes, we adopt the following definition of objects first, as proposed by Bailie, et al.:

> ... an objects first approach ... starts with classes and objects from the first day and introduces basic programming constructs such as conditionals and loops as needed to solve specific problems. [3]

A typical objects first CS1 course would include coverage of such topics as inheritance and polymorphism, perhaps at the expense of other fundamentals like selection or iteration. Philosophically, we believe that teaching more advanced OO concepts early in CS1 would be very difficult, and even counter-productive, for our students. Even objects first proponents agree that teaching CS1 in this manner is challenging [5]. Our IST 145 course *uses* objects from early on, but pushes the writing of classes to the end of the course.

We introduce the use of objects after mathematical operators and expressions, but before coverage of selection and iteration. The nature of Processing requires us to teach how to write methods early on, and so we teach how to write several of the "built-in" methods of Processing (such as the *draw()* method) in the same unit where objects are introduced. We cover selection and iteration after the objects unit. Following iteration, we return to methods, and teach how to write non-built-in methods, including how to use a top-down design approach to create hierarchies of methods. Next, we teach single-dimensional arrays, and basic searching and sorting. Finally, we spend some time learning how to write user-defined classes. More advanced object-oriented design concepts, such as inheritance and polymorphism, are covered in our CS2 course, IST 146.

We find that this approach serves our students well, given that not all of the students in IST 145 are destined to be CS / IS majors. Students taking the course as a mathematics or physics cognate are more likely to require programming to write scripts or short, one-off programs, rather than large, complicated software systems. If we taught the course in a more objects first manner, with emphasis on classes, polymorphism, inheritance, and so on, we would be cheating these students out of concepts they will find useful later. Further, we believe that it is beneficial for CS / IS students to have a firm grasp of selection, iteration, and procedural programming concepts before moving into more fully realized coverage of object oriented programming.

Throughout the IST 145 course, we attempt to emphasize problem solving skills over comprehensive coverage of the details of the programming language. Introductory programming students often have meager problem solving skills (see, for example, Beaubouef, et al. [4]), and so it is important to take such an approach. We employ a Recognize / Analyze / Design / Implement / Support (RADIS) problem solving framework throughout the course. We perform thorough analysis and design steps for each of the examples we teach, employing flowcharts, pseudocode, data dictionaries, and, for the writing classes unit, UML class diagrams. The intent is to teach problem solving, and how to program in general, rather than to dwell deeply on the syntax and semantics of the programming language. We often direct students to the online Processing documentation [10] to learn the details of specific features of the language.

This approach was, in fact, our pedagogy before we decided to adopt Processing as the IST 145 language. Previously, we used Java, and a typical Java programming text, *Java 6 Illuminated* [2], in our IST 145 and IST 146 courses. We adopted Processing for our IST 145 course to try to make programming fun again for our students.

# 3 Rationale for a "Non-Traditional" CS1 Language

Briefly stated, the goal of our IST 145 course is to help students learn how to program, using a language that they find more engaging than traditional languages. As time passes, we hope that this approach will lead to better retention of our CS / IS majors, and that we will also be able to attract more students into our majors or minor.

Although this seems to be slowly changing, introductory programming is traditionally taught in a "console-based" environment, where the user interacts with a program using a text-only interface. A teletype provides a dated but still useful analogy, as does an old DOS-based PC. With a console-based application, the program displays lines of text on the screen, and when the user wants to provide input for the program, the data is entered via the keyboard. Console-based applications provide none of the graphical user interface (GUI) controls we are used to using in our daily work with computers; no buttons, list boxes, windows, and so on. In other words, console-based programs look like they could execute on a teletype, rather than in Windows or Mac OS X.

In traditional introductory programming courses, students typically create console-based applications because they are much easier to write than GUI programs. Unfortunately, console-based programs do not "grab" or excite today's students, who have never used anything but GUI applications. Teaching programming using console-based applications is something akin to teaching teens how to drive a car by driving Model Ts. The fundamental concepts are the same, but the learners are likely to be distracted by the "antique" nature of their vehicle, instead of focusing on learning how to drive.

There have been several theories proposed for why there seems to be low retention in CS / IS majors. For example, Robins, et al. [11] point to the difficulty students have understanding the purpose of programs and their relationship with the computer, while Kinnunen and Malami [7] find that motivation is a major reason for students dropping out. Changing from a traditional, console-based programming environment / language, to some environment that is more graphical and more exciting, is one way that CS educators are trying to address such issues.

Multiple options are available for instructors who wish to use a non-traditional language in their introductory programming courses, including Alice [1], Greenfoot [6], Scratch [12], and Processing. We have found that Processing has several features that make it especially attractive for our IST 145 course.

# 4 Advantages of Processing

Processing is a freely-available language derived from Java. The language and environment are available for free download, in Windows, OS X, or Linux versions. The language is designed to allow programmers to easily add graphics, animation, sound, and user interaction to their programs. Processing provides an exciting, level-appropriate environment for in-

troductory programming, particularly at a liberal arts college. According to the Processing Web site,

> Processing is an open source programming language and environment for people who want to program images, animation, and interactions. It is used by students, artists, designers, researchers, and hobbyists for learning, prototyping, and production. It is created to teach fundamentals of computer programming within a visual context and to serve as a software sketchbook and professional production tool. Processing is an alternative to proprietary software tools in the same domain. [10]

Several aspects of Processing make it very appealing for a CS1 course. First, Processing is freely available for several different computing platforms. Processing programs are created via a simple, uncluttered integrated development environment (IDE). The Processing environment supports three levels of operation, each with a different level of complexity, so concepts can be "scaled up" as the course continues. The language makes graphics and sound easy to access for novice programmers. Processing programs are event-driven, so introductory programming students can carry this concept forward into other classes where we teach true GUI programming. Processing is Java-based, so the transition from Processing to Java in CS2 is relatively easy. The Processing environment has a "one-click" application export option, so students can easily share their work with a larger audience. Processing also has many convenience methods that allow CS1 students to tackle advanced problems without becoming too bogged down in details.

## 4.1 Free and Multi-Platform

Befitting its Java origins, Processing is freely available from the Processing Web site [10], and it runs on Windows, OS X, and Linux systems. The IDE looks and feels virtually identical between the three versions, and, importantly, the keyboard shortcuts are customized between the versions to match the expectations of the operating system. For instance, the shortcut to auto-format code is *control+T* on Windows and Linux systems, but *command+T* on OS X. The Windows download is available in two versions, one for systems that already have a Java Development Kit (JDK) installed, and another for systems without the JDK. Students can run Processing on the system of their choice, and their applications will be completely portable between the different platforms.

## 4.2 Simple IDE

Processing programs, also known as "sketches," are text files, just like programs written in traditional languages. Processing comes with a simple, uncluttered IDE that is used to edit, save, compile, and run sketches. When a sketch runs, a display window appears, showing the output of the sketch. A screen capture of the Processing IDE, and a sample display window, are shown in Figure 1.

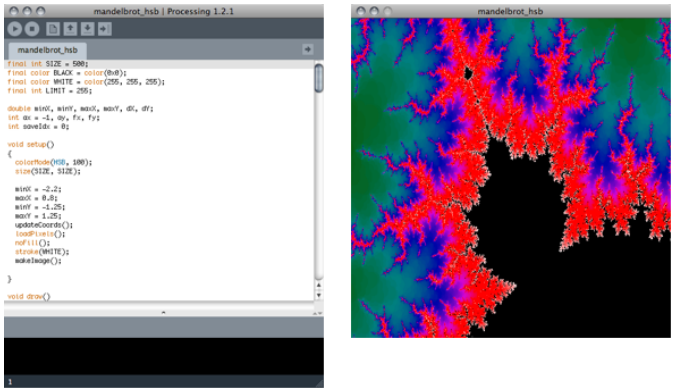


Figure 1: Processing IDE and display window.

The Processing IDE supports basic text editing functions, syntax coloring, and auto-format, which help CS1 students learn the syntax of the language, as well as good coding style. The IDE has a "play" button that compiles and runs the current sketch. The IDE will automatically, through menu options, place import statements at the top of sketches when they are required; for example, when OpenGL rendering is desired. The IDE also provides several useful tools, such as font creation and color choosing. Finally, the IDE has a console area, where syntax errors and runtime exception messages are displayed.

Importantly, the Processing IDE does not present the user with *too much* functionality. The interface is relatively simple, and students are not presented with a sea of toolbar buttons for functions they will never use in a CS1 course. Students moving on into other courses will want more functionality in their authoring tools, but for CS1, the simplicity of the Processing IDE is an asset.

## 4.3 Modes of Operation

Processing supports three modes of operation: *basic*, *continuous*, and *Java*. Basic mode is used for very simple sketches that draw static images. In basic mode, statements are executed in sequence from the top of the sketch to the bottom. Iteration and selection may be used, and built-in methods may be called, but basic mode does not allow programmers to write methods. Basic mode is ideal for learning mathematical expressions and for simple drawing commands. We use basic mode for mathematical operators and expressions, and to introduce the concept of objects and method calls.

Continuous mode is used for animated images or for sketches that involve user interaction, or if user-defined methods are required. A series of built-in methods can be customized to create more complicated applications. For instance, code in the *setup()* method is used to size the application window, load image, sound, or font files, and to configure other aspects of the application. The *draw()* method is called every time the display window is refreshed; code to render the contents of the display window is placed there. Other methods are associated with user interaction events. Programmers may also write other, non-built-in

methods, as needed, for the demands of their applications. Processing and / or Java classes may also be written and used by sketches in continuous mode. We introduce continuous mode when we cover selection, as part of dealing with user interaction. We then use continuous mode through the rest of the course.

Java mode allows complete Java programs to be written in the Processing IDE. In this case, the main class must extend the central Processing class, *PApplet*. We do not use Java mode in our IST 145 course, although we are considering using it to a certain degree in our CS2 course, IST 146.

## 4.4 Graphics and Sound

The primary output of a Processing sketch is an application window, known as the *display window*. The sketch is responsible for drawing the contents of the window, with a variety of method calls for drawing points, lines, shapes, and / or images. In basic mode, a sequence of statements is used to render the contents of the display window. In continuous mode, the contents of the display window are rendered mainly in the *draw()* method, which is called every time the window is refreshed.

Numerous, easy-to-use graphics methods are included in Processing, and rendering can be done in two or three dimensions. Sample graphics methods include the following.

- The *background()*, *fill()*, and *stroke()* methods are used to set the background, fill, and line colors for drawing
- The *point()* method is used to draw a single point in two or three dimensions
- The *line()* method is used to draw a line segment in two or three dimensions
- The *rect()* method is used to draw filled or unfilled rectangles in two dimensions
- The *ellipse()* method is used to draw filled or unfilled ellipses in two dimensions
- The *sphere()* method is used to draw spheres in three dimensions
- The *box()* method is used to draw cubes in three dimensions
- The *image()* method is used to display an image, in JPEG or one of several other common formats, on the screen

These are only a few of the graphics capabilities of the language. For example, Processing also supports typographic output in the display window, using virtually any font imaginable. There are several built-in image filters, like grayscale or blur, that can be applied to pictures. Also, the display window contents can easily be saved, either as static images or video files.

Through the included *Minim* library, Processing sketches can also incorporate music and sound. MP3 and numerous other file formats are supported. Basic playback only takes two

lines of code – one to load the sound file, and one to play it. Other capabilities are available, too, that allow more in-depth manipulation of sound files. For example, once a sound file has been loaded, the programmer can access the actual samples in the file. With that data, graphical visualizations of the sound data can be created, or filters can be applied to the sound before it is played.

The graphics and sound aspects of Processing are the primary features that make the language so attractive to CS1 students. Students especially enjoy being able to incorporate their own image and music files into their programs. Also, when the output of a program is a rendered display window, instead of text output, students are able to more easily associate their program structures with the output of the program.

## 4.5 Event-Driven

By their nature, Processing sketches are event-driven. The *setup()* method is called once, when the sketch starts running, and then never again. The *draw()* method is called every time the display window is refreshed; the frame rate for this depends on the complexity of the output, and the hardware. Other methods are called according to user interactions. Two examples of this type of method are listed here.

- The *mouseClicked()* method is called when the user clicks any one of the mouse's buttons while the display window has focus and the pointer is over the display window
- The *keyPressed()* method is called when the user presses any key on the keyboard while the display window has focus

Built-in variables contain data that can be used in conjunction with these methods to create interactive progams. For instance, the coordinates of the mouse pointer are always contained in the *mouseX* and *mouseY* variables, while the character value of the last key pressed by the user is contained in the *key* variable.

We first introduce some of these methods when we cover objects, before we begin selection or iteration. For instance, we present a single-song MP3 player example (see Section 5.1, below). Examples like this illustrate how Processing applications can be interactive, even without selection or iteration, and also illustrate event-driven programming in a tangible manner.

Using Processing, students naturally and easily grasp the basics of event-driven programming, making it easier for them to progress to traditional GUI programming in follow-on classes.

## 4.6 Java-Based

Another benefit of Processing that carries on into later programming classes is its Java lineage. Processing really *is* Java. Processing sketches are turned into subclasses of the

Processing applet, *PApplet*, automatically, by the IDE, when they are compiled. Built-in methods written by the Processing programmer, such as *setup()* and *draw()*, are actually overrides of methods that already exist in the *PApplet* class. Of course, CS1 students do not need to know that at the time!

Processing's Java roots are most valuable in terms of the basics of programming: Expressions, operators, control structures, and methods. With some minor exceptions, these are all the same between Processing and Java. So, unlike other non-traditional languages such as Scratch or Alice, the syntax that CS1 students learn in Processing can be carried directly over into their other programming courses.

This carry-over breaks down a bit, however, when it comes to object-oriented programming, particularly in terms of information hiding. It is possible to write classes in Processing, but when they are compiled, they become *inner classes* of the main Processing sketch, which is an extension of *PApplet*. In that situation, marking fields and / or methods as "private" has no effect. Members may be labeled as private in Processing classes, syntactically, but doing so does not prevent access by outside entities. The private members of inner classes are directly visible to the enclosing class, so the main program can access private members without causing any syntax errors. Likewise, methods in user-defined classes can access the fields and methods of the main *PApplet* subclass directly. However, since we introduce the writing of classes late in our IST 145 class, it is possible for us to push off full coverage of access modifiers into our CS 2 class, IST 146.

## 4.7 One-Click Application Export and Present Mode

Processing provides easy-to-use facilities that allow students to present and share their work. *Present mode* is accessed through a menu option on the IDE. When present mode is invoked, the sketch in the IDE is compiled and begins to run, but full-screen rather than in a display window. This can be useful for running sketches without the distraction of background windows, but can also be used to write genuine full-screen sketches. For example, a full-screen game could be written to take advantage of present mode.

The IDE also allows users to export their sketches as applications for Windows, OS X, and Linux, or as Web pages. This feature is accessed through either a menu option or a toolbar button on the IDE. The Web export function creates a template Web page and associated JAR files, so that the sketch can be easily published on the Web. The application export option creates double-clickable applications, for any of the supported operating systems. It does not matter which platform the Processing IDE is running on; all three types of applications can be created from any platform.

## 4.8 Convenience Methods

In addition to methods for graphics, sound, and animation, Processing also has other convenience methods that simplify common programming tasks. Many of these can be used by novice programmers to solve problems that might otherwise be unaccessible to CS1

students. Two examples are the *map()* method and the *subset()* method.

The *map()* method is used to map a number from one range to another. This method is especially useful when translating a value from a given range to screen coordinates. The *map()* method can be used, for instance, to convert a float in the range $[-50, 150]$ to a vertical screen coordinate in the range $[0, height)$ (Processing has built-in variables containing the width and height of the display window, named *width* and *height*). For example, the *map()* method could be used in this way to create a bar chart application that draws bars corresponding to a series of Fahrenheit temperatures.

The *subset()* method is one of several methods related to arrays. The *subset()* method is used to extract a sub-array from an existing array. This method was very useful for students when they completed the image recovery project (see Section 5.4, below), for instance. Other array methods exist to copy, reverse, shorten, and lengthen arrays.

These, and other methods available in Processing, allow CS1 students to solve problems that might not be possible in CS1 using more traditional languages. It should be noted, however, that we don't explicitly teach many of these methods. Rather, we refer students to the online documentation and encourage them to discover the capabilities of the language on their own.

# 5 Examples, Assignments, and Projects

The characteristics of Processing listed in the previous section allow CS1 instructors to illustrate programming concepts with examples, assignments, and projects that students find much more exciting and engaging than those associated with more traditional languages. The sections that follow briefly describe some of the examples, assignments, and projects we have used in our Processing-based CS1 course.

## 5.1 Examples: MP3 Players

Since MP3 players are something all our CS1 students can relate to, we use MP3 player examples at two points in the course. First, we use a simple, one-song MP3 player when we discuss objects. Objects are introduced in the third unit of our course, after an introductory unit and one focusing on types, expressions, and operators. This unit is also where we begin to teach event-driven programming, through introductions to the *setup()*, *draw()*, and *mouseClicked()* methods. The example uses three objects: two *PImage* objects to hold "before" and "after" images of Elton John, and an *AudioPlayer* object to hold an Elton John song. When the application runs, the display window shows the "before" image, and waits for the user to click the mouse button. Once the user clicks, the "after" image is shown, and the song starts playing. Screen captures of the MP3 player example are shown in Figure 2.

We return to an MP3 player again in the next unit of the course, which covers selection. The fact that the preceding MP3 player only plays one song provides motivation for the

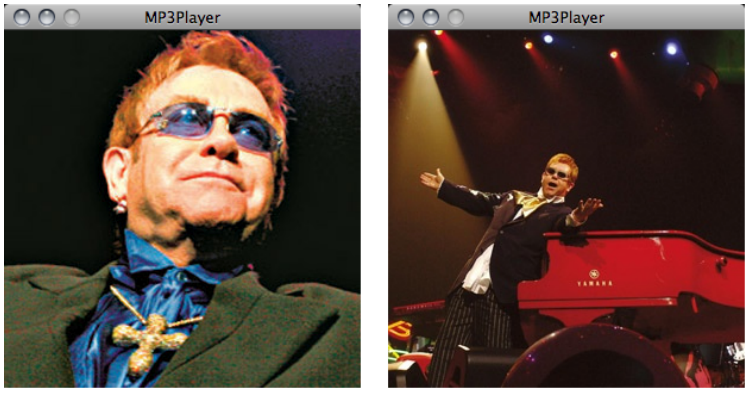


Figure 2: Before and after screencaps of the MP3 player.

selection topic. Once the selection control structures have been described in detail through a series of smaller examples, we present the development of a player that can play four songs instead of one, depending on the user's inputs. Once the user clicks on the display window, nested *if* structures are used to determine which song to play, based on the screen coordinates of the mouse click. A screen capture of the four-song MP3 player is shown in Figure 3.


Figure 3: Four-song MP3 player.

## 5.2 Assignment: Chromakey

One of the assignments we have used involves simple chromakey, or "green-screen," filters. In this assignment, the students are given two images. One is the foreground image, of a cartoon character in front of a green background; the second is the replacement background image. The students have to iterate through the pixels of the foreground image, replacing each green screen pixel with the corresponding pixel from the background image. The new pixels are saved into a third image, which is then written to a file at the end of the algorithm. This assignment can be used in the iteration unit of the course, where a nested *for* loop structure is used to access the pixels in the images. Alternatively, since the pixels in an image may be automatically converted into a linear array, this assignment can be given in the arrays unit, where a single *for* loop can be used to iterate over the arrays of pixels. Before and after chromakey images are shown in Figure 4.

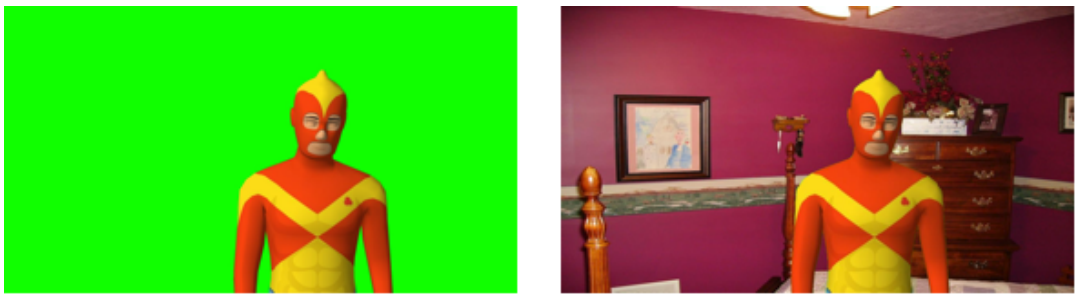

Figure 4: Before and after chromakey images.

## 5.3 Project: Pong

Each time we teach IST 145, we incorporate a semester project, which students complete in small groups. When we assign a semester project, we provide specifications for the minimal functionality that the application must have; teams are encouraged to compete with each other to add additional, creative features.

One of the projects we have used is a re-imagining of the classic video game, Pong. In the project, the right paddle is controlled by the user's mouse movement, while the left paddle must be controlled by a computer player. Also, the application must play an appropriate sound effect each time the ball bounces.

The project involves using objects, detecting collisions with selection, and simple mathematical operators to change the ball's direction and velocity. Therefore, the project can be assigned relatively early in the course, allowing the students plenty of time to finish. Students have an opportunity to add features in several areas. First, we do not specify what algorithm should be used to control the computer paddle; students find it challenging, but achievable, to come up with an "AI" player that is good, but not *too* good. In addition, students can place background images on the screen, to replace the black background, and add looping music to the application. A screen capture of a simple Processing Pong game is shown in Figure 5.

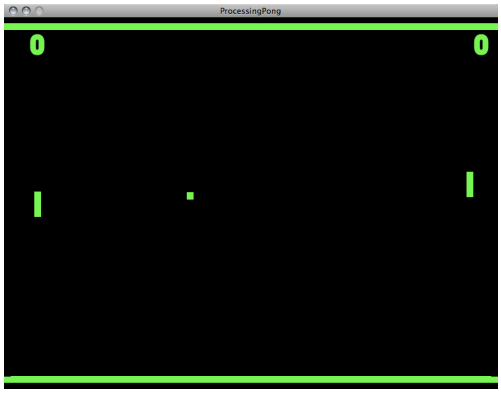


Figure 5: Simple Pong game in Processing.

## 5.4 Project: Image Recovery

This semester project was suggested by David J. Malan in the "Nifty Assignments" session at SIGCSE 2010 [9]. We take a series of photos around the Doane campus with a digital camera, being sure to save them on a freshly-formatted memory card. Using Unix / Linux / OS X tools, we create a "raw" image of the memory card. We then provide the disk image to the students at the beginning of the project. We tell the students that the pictures have been lost, and it is their job to scan through the disk image and recover them. Due to the file format of JPEG files, students can search through the disk image, byte by byte, and easily recognize the beginning of each picture in the file. Then, they only have to extract sub-arrays representing each image, make *PImage* objects from the bytes, and save the results. The students are required to do two further activities. First, the students are required to go on a "scavenger hunt," where they find the location where each photo was taken and take their own version of the shot. Second, the students must do something artistic with the original photos, using only Processing.

This project involves objects, arrays, and iteration, so it must be tackled later in the semester than the Pong project. Due to the simplicity of file I/O in Processing, problems with reading the raw image file and writing the output files are not an issue in the project. Students can focus on the algorithm for extracting the images out of the byte array, rather than worrying about I/O details. Also, the project becomes significantly easier if the students thoroughly explore the online documentation, and discover Processing's array convenience methods. One of the original images, and the filtered version produced by one of our programming teams, are shown in Figure 6.

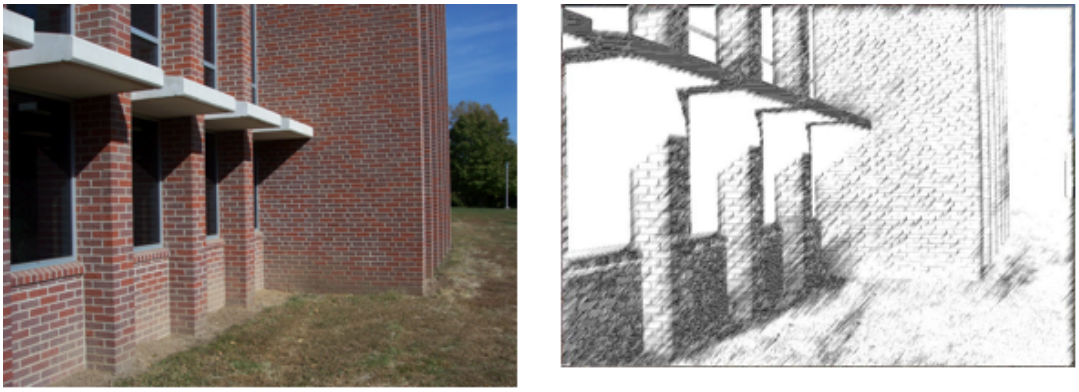

Figure 6: Original and filtered JPEG image.

# 6 Subjective Results

We have only used Processing in IST 145 for two years, and we have no objective data on the impact using Processing instead of Java has had on our students. However, subjectively, we feel that using Processing has been successful and very well received by our students.

The following are a few observations on the use of Processing in the course.

One of the best subjective indicators of student engagement can be seen during closed labs, which we call "hands-on labs" in our course. In these labs, students must solve simple problems related to the current unit, within the time constraints of a single class period. The labs generally involve modifying or completing code provided for them, and the students do not know the assignment before the lab begins. When we used Java in IST 145, students who completed their labs early would simply check out with the instructor and then leave for the day. With Processing, we have seen many students complete their labs, only to stay through the end of the class, having fun and experimenting with the language. A programming class where students do not want to leave at the end of a lab, *because they are having too much fun*, is a successful class.

One of the main motivating factors for our students seems to be that they like using their own materials in their programs. In assignments involving images and sounds, we generally allow the students to use their own – within the limits of good taste, of course. The students enjoy this creative license; it lets them put more of their own personality into their work, which increases ownership in their programs.

Our students have been motivated to learn more about the language, on their own. We refer them to the online documentation frequently, and we make sure that we provide "opportunities" where they *need* to refer to the documentation in order to solve certain problems. The Processing documentation is considerably simpler than the online Java API documentation [8], which contributes to students' willingness to do their own research. Learning how to learn about programming is one of the chief goals of our CS1 course, so we are excited about this trend.

We use Java in our CS2 course, IST 146. While Processing in general prepares students well for IST 146, a drawback we have noticed is that our CS / IS majors wish their IST 146 examples, assignments, and projects were exciting and fun like those in IST 145. Going from the graphical environment of Processing back to console-based Java has been something of a shock for some students, even though we eventually do GUI programming in IST 146. We hope to mitigate this to a certain degree by incorporating some Java-mode Processing work in IST 146, although we have not done so yet.

# 7 Conclusion

We have been very satisfied with our choice to use Processing as the programming language in our CS1 course. The language definitely increases the motivation and degree of engagement of our students, while at the same time providing a solid Java foundation that they carry into later programming courses. We plan to continue to use Processing in IST 145, and we hope to carry over use of the Processing libraries into at least some of the examples, assignments, and projects in our CS2 course, IST 146.

# References


[1] ALICE. Alice. http://www.alice.org/, March 2011.

[2] ANDERSON, J., AND FRANCESCHI, H. *Java 6 Illuminated: An Active Learning Approach*, 2nd ed. Jones and Bartlett, 2008.

[3] BAILIE, F., COURTNEY, M., MURRAY, K., SCHIAFFINO, R., AND TUOHY, S. Objects first - does it work? *J. Comput. Small Coll. 19*, 2 (December 2003), 303–305.

[4] BEAUBOUEF, T., LUCAS, R., AND HOWATT, J. The unlock system: enhancing problem solving skills in cs-1 students. *SIGCSE Bull. 33*, 2 (June 2001), 43–46.

[5] BRUCE, K. B. Controversy on how to teach cs 1: a discussion on the sigcse-members mailing list. *SIGCSE Bull. 37*, 2 (June 2005), 111–117.

[6] GREENFOOT. Greenfoot. http://www.greenfoot.org/, March 2011.

[7] KINNUNEN, P., AND MALMI, L. Why students drop out cs1 course? In *Proceedings of the second international workshop on Computing education research* (New York, NY, USA, 2006), ICER '06, ACM, pp. 97–108.

[8] ORACLE. Java platform se 6. http://download.oracle.com/javase/6/docs/api/, March 2011.

[9] PARLANTE, N., ZELENSKI, J., DODDS, Z., VONNEGUT, W., MALAN, D. J., MURTAGH, T. P., NELLER, T. W., SHERRIFF, M., AND ZINGARO, D. Nifty assignments. In *Proceedings of the 41st ACM technical symposium on Computer science education* (New York, NY, USA, 2010), SIGCSE '10, ACM, pp. 478–479.

[10] PROCESSING. Processing.org. http://www.processing.org/, March 2011.

[11] ROBINS, A., ROUNTREE, J., AND ROUNTREE, N. Learning and teaching programming: A review and discussion. *Computer Science Education 13*, 2 (2003), 137–172.

[12] SCRATCH. Scratch. http://scratch.mit.edu/, March 2011.