IMAGELAB – A PLATFORM FOR IMAGE MANIPULATION ASSIGNMENTS

as published in The Journal of Computing Sciences in Colleges, Vol. 20, Number 1

Aaron J. Gordon
Computer Science Department
Fort Lewis College
1000 Rim Drive
Durango, Co. 81301
970-247-7436
gordon_a@fortlewis.edu

ABSTRACT

This paper describes an image-processing platform called ImageLab. ImageLab provides the infrastructure to allow students to experiment with image manipulation in Java, practice using two-dimensional arrays, and follow specifications to develop a class that is part of a much larger program.

INTRODUCTION

More and more we find that students are motivated by applicable programming assignments. Programming for programming's sake (for example, writing a sort program or a program that only contains linked list manipulation methods) is fun for some students, but is meaningless work for others (especially women [1]). These other students want to see that there are meaningful applications for what they are learning. Described in this paper, the image-manipulation platform ImageLab is a small step in this direction.

ImageLab, written in Java, is a platform for students to write image filtering and manipulation objects. The ImageLab core builds and displays a GUI with menu items to open and save image files. It also has a menu of available filters that can be applied to images. The GUI creates the filter menu dynamically when ImageLab starts up. Without changing ImageLab's code, students can create their own filter objects and have them listed in the filter menu.

STRUCTURE

The ImageLab classes provide methods that input and display images. The students can then write image filters that modify these images.

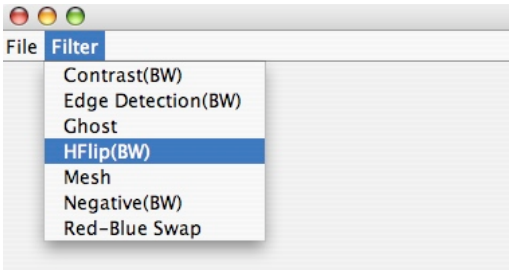
ImageLab starts by creating menu items for each available filter and instantiating the filter objects. These filter objects are classes that implement the ImageFilter interface. When the user chooses an image by clicking on the open menu item, ImageLab opens and displays the image. This image is then available for filtering.

When the user chooses a specific filter from the filters menu, that filter is applied to the image. To make a new filter available, students must have their filter class implement the ImageFilter interface and they must store their .class file in the appropriate directory.

Examples Of Use:

Here is a picture of two bison facing to the image's left.

The user might then use the ImageFilter menu to choose a filter.



This filter flips the image resulting in the following image with the bison facing to the image's right:

The filter supplies both the text appearing in the filter menu as well as the title of the resulting image. The filters shown are not part of ImageLab's core. Students develop these filters as plug-ins to ImageLab. The code for the filter used above is shown later in this paper.

The ImageFilter Interface

ImageLab is written in Java. All filters, developed as ImageLab plug-ins, must implement the ImageFilter interface. The ImageFilter Interface defines three methods.

```
public void filter(ImgProvider ip)
```

The filter method is called by ImageLab when the corresponding menu item is chosen. ImageLab passes in an ImgProvider object (described in the next section) that is responsible for the current image. The ImgProvider object can supply the image in color or in black and white. It stores the image in RGBA format.

The filter method, after it has manipulated the object, creates a new ImgProvider object to store the modified image. This ImgProvider object can then be asked to display the image with a title passed in by the ImageFilter object.

The second method defined in the ImageFilter interface is:

```
public ImgProvider getImgProvider();
```

This method returns the filtered image to the caller. One time this call occurs is when the user chooses `save` from the `file` menu. ImageLab, acting as the menu item's ActionListener, retrieves the ImgProvider object from the filter and stores the image in a file.

The final method defined in the ImageFilter interface is:

```
public String getMenuLabel();
```

This method returns the String to be used as this filter's menu label. When building the filters menu, ImageLab calls getMenuLabel() for each available filter.

ImgProvider

Each object from the ImgProvider class is responsible for one image. ImgProvider stores the image in an array of `int` where each element represents one pixel in RGBA format. The object also holds the information for each of the four channels (red, green, blue, and alpha) in individual two-dimensional arrays. An ImgProvider object has methods to return any of the four channels and can also return a two-dimensional array holding a gray-scale representation of the image. In addition, ImgProvider supplies file I/O methods to read and write images, as well as methods to display images.

Example Filters

This first example is a filter that horizontally flips a black and white image.

```
package filters;
import imagelab.*;
public class HFlip implements ImageFilter {
    ImgProvider filteredImage;   //to hold the modified image
public void filter (ImgProvider ip) {
   short tmp;
A  short[][] im      = ip.getBWImage();  //gets b&w image
   int height  = im.length;
   int width   = im[0].length;
B  for (int r = 0; r<height; r++) {  //invert each row
        for (int c=0, x = width - 1; c < x; c++, x--) {
            tmp = im[r][c];
            im[r][c] = im[r][x];
            im[r][x] = tmp;
        }//for c
    }//for r;
C  filteredImage = new ImgProvider();
   filteredImage.setBWImage(im);
   filteredImage.showPix("Flipped Horizontally");
}//filter

public ImgProvider getImgProvider() {  //returns the modified image
    return filteredImage;
}//getImgProvider
public String getMenuLabel() { //return label for filter menu
    return "HFlip(BW)";
} //getMenuLabel
}
```

At the line marked **A**, the filter method accesses the image in black and white (as a gray scale). At the nested for-loops (marked **B**) each row is reversed. The final three lines of the filter method (marked **C**) store the resulting image in an ImgProvider object and display the image to the user.

This next sample filter creates a mesh effect with the image; turning the image into a checkerboard with all of the white squares transparent. The size of each square is selected by the user at run time.

```
package filters;
import imagelab.*;
import javax.swing.JOptionPane;

public class Mesh implements ImageFilter {

    ImgProvider filteredImage;   //to hold the modified image

public void filter (ImgProvider ip) {
    short tmp;
A   short [][] al = ip.getAlpha();  //alpha from original picture
    int npixels   = getMeshSize();  //holds size of each mesh square
B   for (int r = 0; r<al.length; r++) {
        for (int c=0; c<al[0].length; c++) { //set some alpha to 0
            if ((c/npixels+r/npixels) % 2 == 1) al[r][c] = 0;
        }//for c
    }//for r;
C   filteredImage = new ImgProvider();
    filteredImage.setColors(ip.getRed(), ip.getGreen(),
                                              ip.getBlue(), al);
    filteredImage.showPix("Transparent");
}//filter

public ImgProvider getImgProvider() {
    return filteredImage;
}//getImgProvider

public String getMenuLabel() {
    return "Mesh";
} //getMenuLabel

D  protected int getMeshSize() {//returns retrieved mesh size
    String response = JOptionPane.showInputDialog(null,
                                              "Enter Mesh Size", "");
    int num;
    try {
        num = Integer.parseInt(response);
        if (num <= 0) num = 2;
    } catch (NumberFormatException ex) {
        num = 2;
    }//catch
    return num;
}//getMeshSize
}
```

At point **A**, the Mesh object obtains the image's alpha channel (which controls transparency) as a two-dimensional array. The user inputs the mesh square size in the method `getMeshSize()` (point **D**). The nested for loops (point **B**) change the alpha value to zero (meaning completely transparent) for specific pixels. And, finally, the changed image is stored in an ImgProvider object and displayed (point **C**).

Here is the bison image after being filtered with a mesh size of four. Notice that the black background shows through the image.

INTENDED AUDIENCE

ImageFilter is useful toward the end of CS1 and/or the beginning of CS2. Assignments can be as simple and straightforward as the horizontal flip filter above or can be open ended as in having the students write a non-trivial filter of their own.

Writing filters gives students practice in using two-dimensional arrays, object-oriented programming, interfaces, reading APIs, and writing according to specifications. In addition, it gives students a chance to be creative and gives them immediate visual feedback on the results of their program.

REFERENCES

[1] Jane Margolis and Allan Fisher, *Unlocking the Clubhouse, Women in Computing*, MIT Press, 2002