TEACHING FILE INPUT/OUTPUT, LOOPS, AND IF-STATEMENTS VIA A RED EYE REDUCTION ASSIGNMENT

Timothy Urness
Department of Mathematics and Computer Science
Drake University
Des Moines, IA 50311
515-271-2118
timothy.urness@drake.edu

ABSTRACT

This paper describes a "nifty" programming assignment that requires students to use files, loops, and if-statements to implement an algorithm that will remove the red-eye artifact from an image. The assignment is most suitable for a CS1 course, but could be altered to accommodate a CS0 or CS2 course.

1 INTRODUCTION – WHY IS THE ASSIGNMENT "NIFTY"?

The red-eye artifact in photographs occurs when the light from a camera's flash reflects off the retinas of the subject. As a result, the pupils of the subject often appear red or have a red halo in the vicinity [1]. Obviously, the effect is undesirable as it makes the subject appear unnatural.

The application of removing red-eye artifacts from images is undoubtedly familiar to most students. Therefore, engaging students in developing an elementary form of this program is relevant to the types of computing students encounter on a daily basis.

There are many professional software programs that will reduce or eliminate the red-eye artifact from images [1]. While students in a CS1 course may not yet have the skills to develop a sophisticated algorithm, the exercise can be used to motivate the usefulness of file input and output, loops, and if-statements. The author has found this programming assignment particularly useful in inspiring students while motivating the importance of algorithm development.

2 THE ASSIGNMENT

2.1 The Basic Algorithm

An elementary version of a red-eye reduction algorithm can be accomplished as follows: open an image file, use a loop to scan through all pixel values, use an if-statement to determine if a pixel is "red," change the "red" pixel, write the pixel values to another file. Note that the reading of the input file and the writing to the output file can be done inside the loop, so there is no need to store the image in a data structure.

The basic algorithm will simply change all "red" pixel values to another color – regardless of their location within an image. A more descriptive name for this simplified program would be a *red reduction* algorithm, rather than a *red-eye reduction* algorithm. However, if students are supplied with an image in which the only "red" pixels are located in the eye, running this algorithm will result in a dramatic improvement of the image. Figure 1 is an example of an image on which the simplified *red reduction* algorithm works as effectively as a *red-eye reduction* algorithm. Most importantly,

students get an opportunity to implement file input and output, loops, and if-statements in the context of a “real-world” application.

Figure 1: An image that suffers from the red-eye artifact.

2.2 Algorithm Details

Pixels stored in an image have three color components: red, green, and blue. In the simplest case, each component is stored as an integer that ranges from 0 to 255. A pixel colored black has the values red = 0, green = 0, blue = 0. A pixel colored white has the values red = 255, green = 255, blue = 255. All other possible colors can be represented by a combination of the values of these components. For the image in Figure 1, a “red” pixel is defined as a pixel in which the red component is greater than the sum of the green and blue components. If the pixel is “red,” then the red component is changed to match the blue component. This algorithm results in the image depicted in Figure 2. While this will not work for most images that contain the red-eye artifact, it works reasonably well for a few select images.

Figure 2: The result of applying the simple red reduction algorithm to the image in Figure 1.

3 PROJECT DETAILS

While the algorithm itself is rather simple, there are several details that must be thoroughly explained in order to make the assignment manageable for a CS1 student. These issues include how the image file is stored and how to view an image.

3.1 The PPM file format

The PPM (Portable Pixel Map) file format is a simple image file format. There are two different formats for PPM images: binary and ASCII. When stored in the ASCII format, the PPM format is a simple way to store an image that is easily readable and writeable. A PPM file begins with the line that identifies it as an ASCII PPM image file:

```
P3
```

The second line of the file is a comment that begins with the # character, such as

```
# This is an example of a comment for the PPM file format
```

The third line of the file contains two integers: the width of the image, followed by a space, followed by the height of the image. An image that is 100 pixels wide and 120 pixels tall would have the line

```
100 120
```

The fourth line of the file is a single integer: the maximum color value for any component of the file. In our example, this number will always be

```
255
```

The fifth (and following) lines of the file begin depicting the image. The fifth line contains an integer between 0 and 255 that is the **red** component for the first pixel. The sixth line contains the **green** component for the first pixel. The seventh line is the **blue** component for the first pixel. The eighth line is the **red** component for the *second* pixel, and so on. For example, the first ten lines of the PPM file for Figure 1 are depicted in Figure 3.

While the pixel value components can also be stored on the same line delineated by spaces, the program is easiest to implement if the file structure is as described above. Lastly, PPM files typically end with a .ppm extension.

```
P3
# CREATOR: The GIMP's PNM Filter Version 1.0
400 374
255
133
127
118
104
105
99
```

Figure 3: The first ten lines of a PPM image file

3.2 Image Viewing Software

After creating an image file, the students must also view it to determine whether their algorithm was effective. Unfortunately, not all image-viewing software will recognize PPM files. If students are programming on their home machines, they may need to download an additional program. GIMP [2] is a free image processing program

that is capable of displaying PPM image files. While there are other programs capable of doing this, such as Adobe Photoshop[3] or irfanview[4], the author has found GIMP to be the most convenient as it is free and can be installed on a variety of operating systems (e.g. Mac, Windows, and Linux).

4 ADDITIONAL OPTIONS FOR THE ASSIGNMENT

In order to challenge students, the program can also incorporate additional tasks. Some suggestions include:

- •Prompt the user for the input and output filename and run the algorithm on these files. Note: the image should be able to be any size.
- •Use nested loops to process the image.
- •Instead of the simplistic red reduction algorithm described previously, develop a more sophisticated and flexible red reduction algorithm.
- •Calculate the maximum and minimum red component of the original image. Print these values to the console.
- •Create a black "frame" around the image that is at least five pixels wide. (See Figure 4).
- •Create a grey-scale image of the original color image. A grey-scale image is an image that has the same numeric value for the red, green, and blue components of each pixel. Calculate the average (integer) value for each individual pixel, and use this value to create the grey-scale pixel. Give the user the option of applying the red-reduction or the grey-scale algorithm when using your program. (See Figure 4).

Figure 4: Additional programming requirements can involve turning a color image to a grey-scale image and/or putting a black border around the image.

4.1 Adaptations for CS0

As part of the Nifty Assignments for SIGCSE 2007, John Cigas recommended using programming elements of spreadsheets [5]. Pixel information could be stored in a spreadsheet and eventually exported into a PPM format, reducing the amount of programming required of the students.

4.2 Adaptations for CS2 or Data Structures

The project described could be made as complicated as the instructor desires. One option is to require students to create their own data structures to store each individual pixel. Another data structure could be created to store the image. A two-dimensional array could also be incorporated, as it is a natural structure for storing images.

CONCLUSION

This paper describes a "nifty" programming assignment that requires students to use files, loops, and if-statements to implement an algorithm that will remove the red-eye artifact from an image. The assignment incorporates a real-world application to enforce the usefulness of standard algorithm constructs typically introduced in a CS1 course. The assignment could be altered to accommodate a CS0 or CS2 course.

ACKNOWLEDGEMENTS

I would like to acknowledge Ben Schafer for helping inspire the idea of using graphics-based programming in an introductory course. The cat image used in this paper is used with the permission of Kim I. Martin.

REFERENCES

[1] Zhang, L., Sun, Y., Li, M., Zhang, H., Automated red-eye detection and correction in digital photographs, *International Conference on Image Processing ICIP*, 2363-2366, 2004.

[2] The GNU Image Manipulation Program, http://www.gimp.org/ , retrieved January 10, 2008.

[3] Adobe Photoshop CS3, http://2008-version.com/photoshop/ , retrieved January 10, 2008.

[4] IrfanView – Official Homepage – one of the most popular viewers worldwide, http://www.irfanview.com/ , retrieved January 10, 2008.

[5] Parlante, N. Cigas, J., Shiflet, A., Sooriamurthi, R., Clancy, M.J., Noonan, B., Nifty Assignments, *Proceeding of the Thirty-Eighty SIGCSE Technical Symposium on Computer Science Education*, 497-498, 2007.