Fundamentals of Computer Science I (CS151.02 2007S)

Laboratory: Image Manipulation

Summary: We explore techniques for algorithmically modifying existing images.

Contents:

- Preparation
- Exercises
  - Exercise 1: Getting Colors
  - Exercise 2: Setting Colors
  - Exercise 3: Simple Transformations
  - Exercise 4: Changing Reds
  - Exercise 5: Multiple Transformations
  - Exercise 6: Conditional Manipulation
  - Exercise 7: Anonymous Transformations

Preparation

a. Create a new file for your procedures for this lab (say, `image-manip.scm`). Open that file in DrScheme.

b. Start the GIMP and open a Script-Fu console.

c. Load your new file and `/home/rebelsky/Web/Courses/CS151/2007S/Examples/hog.scm`.

d. If you're working on a computer outside the MathLAN, you may want to download a new copy of `gimp.scm`.

e. For this laboratory, you will need a moderately small image (say 128x128), since the number of points we process is approximately width*height. (A 128x128 image has more than sixteen thousand points.) You can make a copy of the sample image or you can select a portion of some image you find interesting (such as your StalkerNet photo). I'd suggest that you call your image `sample.jpg`.

Exercises

Exercise 1: Getting Colors

a. Load your sample image with

```
(define sample (load-image "/home/username/Desktop/sample.jpg"))
```

b. Display it with

```
(show-image sample)
```

c. Determine the color of your image at the position (10,10) with

```
(color->list (get-color-at sample 10 10))
```

d. Set the foreground color to that color with

```
(set-fgcolor (get-color-at sample 10 10))
```

e. Try getting a few other colors.

Exercise 2: Setting Colors

The reading suggests that you can set colors with the `set-color-at!` procedure, but that you can't necessarily see those changes.

a. Try setting the color at (10,10) to red with

```
(set-color-at! sample 10 10 RED)
```

b. Verify that you've set the color with

```
(color->list (get-color-at sample 10 10))
```

c. Zoom in to 800% or so (using View>Zoom) and see if you can see the change. Don't worry if you can't; it's one of the strange design features of GIMP.

d. Open a new copy of your sample image, set the color at (10,10), and then show it.

```
(define sample (load-image "/home/username/Desktop/sample.jpg"))
(set-color-at! sample 10 10 RED)
(show-image sample)
```

e. Verify that you've set the color with

```
(color->list (get-color-at sample 10 10))
```

f. Zoom in to 800% or so and see if you can see the change. This time, you should be able to see the change. If you can't, get help.

Exercise 3: Simple Transformations

a. Convert your sample image to greyscale with

```
(define sample (load-image "/home/username/Desktop/sample.jpg"))
(modify-image! greyscale-better sample)
(show-image sample)
```

Warning! It takes about a minute for `modify-image!` *to do its job.*

b. Rotate the color components of the original image with

```
(define sample (load-image "/home/username/Desktop/sample.jpg"))
(modify-image! rotate-components sample)
(show-image sample)
```

c. Increment the red component of the original image with

```
(define sample (load-image "/home/username/Desktop/sample.jpg"))
(modify-image! redder-64 sample)
(show-image sample)
```

d. Decrement the blue component of the original image with

```
(define sample (load-image "/home/username/Desktop/sample.jpg"))
(modify-image! lessblue-64 sample)
(show-image sample)
```

Exercise 4: Changing Reds

As you may recall from the reading, you can increment the red component of a color by 64 in a variety of ways.

- `(redder-64 color)`
- `(change-red 64 color)`
- `((redder 64) color)`
- `((l-s change-red 64) color)`

a. Create the color light-grey with

```
(define light-grey (rgb 64 64 64))
```

b. Determine the components of that color with

```
(color->list light-grey)
```

c. Verify that each of the procedures above changes the red component by 64. For example,

```
(color->list (redder-64 light-grey))
```

d. Figure out a way to increment the red component of `light-grey` by 128.

e. What do you expect to happen if you decrement the red component of `light-grey` by 128? Verify your answer experimentally.

f. Create a new copy of your sample image with the red component of each point incremented by 128.

g. Create a new copy of your sample image with the red component of each point set to 0.

Exercise 5: Multiple Transformations

We've seen that there are two ways to sequence image modifications:

- We can simply type the two `modify-image!` commands in sequence.
- We can use one call to `modify-image!` and compose the color transformations.

For example, if we want to both increment the blue component by 96 and decrement the red component by 64, we can write

```
(modify-image! (bluer 96) sample)
(modify-image! (redder -64) sample)
```

or we can write

```
(modify-image! (compose (redder -64) (bluer 96)) sample)
```

a. One way to compare the efficacy of the two approaches is to time them. Try both modifications (on separate copies of the image, preferably), timing each. Which is faster? Are the results the same?

c. At this point, you know a few basic modifications: (1) setting the image to greyscale with `greyscale-better`; (2) changing one of the color components with `(redder amt)` (or `bluer` or `greener`); rotating the color components with `rotate-component`. Try composing some of these.

Exercise 6: Conditional Manipulation

At times, we only want to change color values if certain conditions hold. For example, we might only want to increment the red value if it is below or above a certain amount.

a. In your code file, add the following procedure

```
(define sometimes-change-red
  (lambda (cutoff amt)
     (lambda (color)
       (if (> (red color) cutoff)
           (change-red amt color)
           color))))
```

b. What would you write to increment by 128 the red component of all of the pixels with a red component of at least 64?

c. One hopes that you wrote something like

```
(modify-image! (sometimes-change-red 64 128) sample)
```

Try that code.

Exercise 7: Anonymous Transformations

As you may recall from a previous laboratory, when experimenting with transformations, it may be useful to try some anonymous ones before deciding on a final one.

For example, suppose we want to increment the red component by 25%, the green component by 50%, and the blue component by 75%. We can write

```
(modify-image! (lambda (color) (rgb (* 1.25 (red color)) (* 1.5 (green color)) (* 1.75 (blue color)))) sample)
```

Similarly, if we want to try setting the blue component to 128 and swap the red and green components, we can write:

```
(modify-image! (lambda (color) (rgb (green color) (red color) 128)) sample)
```

a. Verify that these techniques work.

b. Consider the following procedure

```
(define extreme (lambda (val) (if (< val 128) 0 255)))
```

What do you expect to happen with the following?

```
(modify-image! (lambda (color) (rgb (extreme (green color)) (extreme (red color)) (extreme (blue color)))) sample)
```

c. Try it.

d. Try a few interesting modifications of your own.

Copyright © 2007 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-nc/2.5/` or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.