Programming Languages (CS302 2007S)

Laboratory: Higher-Order Graphics

Summary: We explore techniques for algorithmically creating and manipulating grid-based images.

Contents:

Exercises

Exercise 0: Preparation

For the first part of this laboratory, you will need to have a copy of `grid.scm` in your GIMP scripts folder.

a. Open a new terminal window.

b. Type

```
ln -s /home/rebelsky/Web/Courses/CS302/2007S/HOG/grid.scm ~/.gimp-2.2/scripts/grid.scm
```

c. Start GIMP and verify that the Xtns/Script-Fu menu contains a Glimmer submenu.

For the second part of 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 select a portion of some image you find interesting (such as your StalkerNet phot). I'd suggest that you call your image `sample.jpg`.

Exercise 1: Color Grids

a. Set the current brush to a medium circle.

```
(set-brush "Circle (09)")
```

a. From the Xtns menu, select Script-Fu, then Glimmer, and finally `Color Grid`. A dialog box should appear. Enter 100 for the width and height, and 10 for the horizontal and vertical spacing. Observe the image that appears.

b. What do you expect to happen if you use 8 for the horizontal and vertical spacing? Confirm or reject your prediction experimentally.

c. Set the current brush to a larger fuzzy circle.

```
(set-brush "Circle Fuzzy (15)")
```

d. What do you expect to happen if you again use 8 for the horizontal and vertical spacing? Confirm or reject your prediction experimentally.

e. What do you expect to happen if you use `func3` for the red component (continuing to use `func2` for green and `func3` for blue)? Confirm or reject your prediction experimentally.

f. Try one or two other brushes and one or two other arrangements of functions.

Exercise 2: Specifying Components

a. With DrScheme, create a new file on your desktop, `components.scm`.

b. Add the following function to that file.

```
(define func4
  (lambda (x y)
      (modulo (+ (* 3 x) (* 5 y)) 256)))
```

c. Make a link to that file in your Script-Fu scripts folder by opening a new terminal window and typing the following command:

```
ln -s ~/Desktop/components.scm ~/.gimp-2.2/scripts/components.scm
```

(You only need to do this once, but it makes sure that the GIMP knows about the file.)

d. Refresh scripts by selecting Xtns->Script-Fu->Refresh Scripts. (You will need to refresh scripts whenever you change `components.scm`.)

e. Verify that you can now use `func4` in building a Color Grid.

f. Add the following procedure to `components.scm`.

```
(define func5
  (lambda (x y)
      (modulo (* x (abs (sin y))) 256)))
```

g. Verify that you can use this procedure. (If you can't remember that you need to save the file and to refresh scripts.)

h. Add a few of your own "interesting" component functions.

Exercise 3: Anonymous Components

As you may have noted from the previous exercise, it can be a bit of a pain to remember to save and refresh every time you define a new component function. As the reading suggests, we can instead use anonymous functions.

a. Open the Color Grid dialog box, use the following for the red, green, and blue components, and then display the grid.

- `(lambda (x y) (modulo (* x 5) 256))`
- `(lambda (x y) (modulo (* y 5) 256))`
- `(lambda (x y) 0)`

b. Try some functions of your own design. Remember that each function has the form `(lambda (x y)` *`expression`*`)` and that the expression should return a value in the range [0..255].

Exercise 4: Making Grids from the Console

Some people (myself included) prefer to generate their grids from the console. As you may recall from the reading, the `color-grid` procedure has seven parameters:

- the width of the image
- the height of the image
- the horizontal spacing between points
- the vertical spacing between points
- the red component function
- the green component function
- the blue component function

For example, here's the command that we might use for the first grid we drew (a 100x100 image with a 10x10 grid)

```
(color-grid 100 100 10 10 func1 func2 func3)
```

a. Try that example.

b. Try varying the parameters (e.g., using a different width, height, spacing, or function).

c. Try using lambda expressions for the functions. For example

```
(color-grid 100 100 8 9 (lambda (x y) (modulo (* x y) 256)) (lambda (x y) (modulo (* x 5) 256)) (lambda (x y) (trunc (* 255 (abs (sin (* x y)))))))
```

Exercise 5: 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 6: 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 7: 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 8: 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 9: 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 10: 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 11: 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.