

## Homework 12: Color Grids, Revisited

Assigned: Tuesday, April 10, 2007

Due: Friday, April 13, 2007

*No extensions!*

**Summary:** In this assignment, you will further explore the creation of art via algorithm in the GIMP.

**Purposes:** To give you further experience using anonymous and higher-order procedures.

**Expected Time:** One to two hours.

**Collaboration:** You may work in a group of any size between one and four, inclusive. You may consult others outside your group, provided you cite those others. You need only submit one assignment per group.

**Submitting:** Email me your work, using a subject of *CSC151 Homework 12*.

**Warning:** So that this exercise is a learning assignment for everyone, I may spend class time publicly critiquing your work.

## Background

In a recent lab, you experimented with the creation of color grids that used an algorithm to choose the color at each position in the grid. In this assignment, you will rewrite the core color-grid procedure to permit different brushes and different grid spacing.

In case you're wondering, you should not rely on the code I wrote for the `color-grid` procedure, since that code uses some control structures you have not learned yet (in part, because they are specific to Script-Fu). Rather, you will need to rewrite your procedure from scratch.

However, you may find it useful to consider the following procedure, which draws one point in the image.

```
(define draw-one-point
  (lambda (image x y redfunc greenfunc bluefunc)
    (set-fgcolor (list (mod (redfunc x y) 256)
                      (mod (greenfunc x y) 256)
                      (mod (bluefunc x y) 256)))
    (blot image x y)))
```

The `blot` procedure draws a single spot with the current paintbrush and the current foreground color at the specified location.

## Assignment

a. Write a procedure, (`draw-simple-grid image width height redfunc greenfunc bluefunc`) that draws an evenly spaced eleven-by-eleven grid on the image, using the current paintbrush and choosing the color at each position using the three functions. (You may call `draw-one-point` for each point in the grid.) Note that I've chosen an eleven-by-eleven grid because it is relatively easy to compute the points on the grid. The first x value is 0. The next is 10% of the width. The next is 20% of the width. And so on and so forth. The eleventh is 100% of the width. Similarly, the first y value is 0. The next is 10% of the height. And so on and so forth.

b. Grid based drawing can be more interesting when you permit multiple brushes. Extend `draw-simple-grid` to take an additional procedure, `brushfunc`, as a parameter. This new parameter should be a function of two integer parameters, `x` and `y`, and should select a brush name based on `x` or `y` (or both). For example, here's a very simple such procedure:

```
(define simple-brush
  (lambda (x y)
    (if (odd? (+ x y))
        "Circle (09)"
        "Circle Fuzzy (07)")))
```

c. Test your new procedure with a variety of brush functions.

d. Some might criticize the images created by `color-grid` or `draw-simple-grid` for the stunning regularity of the placement of dots. Add two more parameters, `xfun` and `yfun`, each of which takes a number between 0 and 1 (representing how far we are across the "grid") as a parameter and returns a number between 0 and 1 (representing how far across the grid we actually draw the point) as a result. For example, for even spacing, you might use

```
(draw-simple-grid image
  180 100           ; width, height
  func1 func2 func3 ; red, green, blue
  simple-brush     ; brush
  (lambda (x) x)   ; x coordinate
  (lambda (y) y)) ; y coordinate
```

For somewhat more interesting spacing, you might use `(lambda (x) (* x x))` for `xfunc`. In that case, the x positions in the image would be computed as follows

- First x position:  $0^2 = 0$ .  $0 * 180 = 0$ .
- Second x position:  $.1^2 = .01$ .  $.01 * 180 = 1.8$ .
- Third x position:  $.2^2 = .04$ .  $.04 * 180 = 7.2$ .
- ...
- Ninth x position:  $.9^2 = .81$ .  $.81 * 180 = 145.8$

e. Generate three interesting images using your procedure and send me the instructions for generating those images. (Do not clog my inbox with the images themselves.)

## Helpful Hints

1. For the initial version of `draw-simple-grid`, you might want to use a helper that draws a single row of the grid, and then call that helper for each value of `y`.
2. Try to have some fun.
3. Some of you might need help thinking about how you recurse across an image in percents. Here's a procedure that calls `draw-one-point` along the major diagonal of the image, using the same percentage step as suggested above.

```
(define draw-diagonal
  (lambda (image width height redfunc greenfunc bluefunc)
    (letrec ((kernel (lambda (percent)
                      (if (<= percent 1.0)
                          (let ((x (* percent width))
                              (y (* percent height)))
                            (draw-one-point image
                                             x y
                                             redfunc greenfunc bluefunc)
                            (kernel (+ percent .10))))))
      (kernel 0))))
```

Here's a sample call to that procedure:

```
(draw-diagonal img 180 100 (lambda (x y) (* x 2)) (lambda (x y) (* y 3)) (lambda (x y) (+ x y)))
```

4. As many of you have noted, Script-Fu is not a complete Scheme. One thing that is missing is the set of procedures that convert reals to integers, such as `round` and `floor`. Fortunately, `draw-one-point` seems to work fine with reals that include fractional portions, such as 2.4.

---

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.