Fundamentals of Computer Science I: Media Computing (CS151.02 2007F)

Laboratory: Transforming Images

Summary: In this laboratory, you will extend the operations you've used to transform colors into operations that transform images.

Contents:

Reference:

Preparation

In this laboratory, you will be creating a few images and manipulating others. We will also be working with some colors.

Create a new 4x3 image, call it `canvas`, show it, and zoom in to 16x resolution.

Open an existing image of your choice, call it `picture`, and show it. Please choose an image that is not too large (say, not much more than 250x250).

Load your list of favorite colors, `/home/`*`username`*`/Desktop/fave-colors.sct`. If you don't have that list, use `/home/rebelsky/glimmer/samples/fave-colors.sct`.

Exercises

Exercise 1: Mapping Transformations

a. Set a few pixels in `canvas` to colors of your choice. Leave others black or white.

b. What do you expect to happen when you use `image.map!` to complement each pixel in `canvas`, using the following instruction?

```
(image.map! rgb.complement canvas)
```

c. Check your answer experimentally.

d. What do you expect to have happen if you use `image.map!` to complement each pixel in `picture`? (You would use nearly the same instruction, substituting `picture` for `canvas`.)

e. Check your answer experimentally.

f. What do you expect to have happen if you once again complement each pixel in `picture`?

g. Check your answer experimentally.

Exercise 2: Undoing Transformations

a. What do you expect to have happen if you use `image.map!` to redden each pixel in `canvas`?

b. Check your answer experimentally.

c. You may have noticed that in the previous problem, we were able to undo the complement transformation by complementing again. Is there an easy way to undo the redden operation? (You do not have to write code; just explain how to do it.)

d. Are there transformations or sequences of transformations that would be impossible to undo? (That is, can you do something to an image such that there is nothing that you can do to the revised image that will bring back the original image?)

Exercise 3: map vs. map!

As you may have just observed, there are times that transforming an image can be dangerous, because we cannot easily undo some transformations. As an alternative, many programmers build new images that simulate the transformation, rather than transforming the existing in place. In DrFu, the `image.map` operation does just that. It returns the identifier of a new image.

a. Consider the following instruction. What effect do you have expect this instruction to have on `canvas`?

```
(define new-image (image.map! rgb.darker canvas))
```

b. Check your answer experimentally.

c. Consider the following instruction. What effect do you have expect this instruction to have on `canvas`?

```
(define newer-image (image.map rgb.darker canvas))
```

d. Check your answer experimentally.

e. What do you expect to have happen if we show `newer-image`?

f. Check your answer experimentally.

```
(image.show newer-image)
```

g. What do you expect to have happen with each of the following:

```
(define img1 (image.map rgb.rotate canvas))
(define img2 (image.map rgb.lighter canvas))
(define img3 (image.map rgb.phaseshift canvas))
(image.show img1)
(image.show img2)
(image.show img3)
```

h. Check you answer experimentally.

Exercise 4: Composition

Consider the following definitions.

```
(define much-darker (compose rgb.darker rgb.darker))
(define red (rgb.new 255 0 0))
```

a. What color do you expect `(much-darker red)` to compute? (Answer the question in terms of red, green, and blue values.)

b. Check your answer experimentally.

c. Set the top-left pixel of `canvas` to red.

d. What effect do you expect the following instruction to have?

```
(image.transform-pixel! canvas 0 0 (compose rgb.lighter rgb.lighter))
```

e. Check your answer experimentally.

f. What effect do you expect the following instruction to have?

```
(image.transform-pixel! canvas 0 0 (compose rgb.lighter (compose rgb.lighter rgb.lighter)))
```

g. Check your answer experimentally.

Exercise 5: Composition and Order of Operations

Consider the composition `(compose rgb.darker rgb.phaseshift)`.

a. Does this darken the image first or phase shift the image first.

b. Does it matter? That is, do you get the same result either way?

c. Check your answer experimentally with `image.map`.

Exercise 6: Undoing Transformations

Earlier in this lab, we saw that some transformations had natural inverses and some did not. For example, if you complement a color twice, you get the original color. For many colors, if you lighten and then darken the color, you get the original color. (If any of the components is very large, then we may not be able to restore the original color.)

a. Consider the `rgb.redder` operation. How would you write an inverse to that operation using the based color transformations along with composition?

b. Test your answer using `image.map`. For example, if you decided that the answer was to make the image greener and bluer, you might write something like the following.

```
(define redder-canvas (image.map rgb.redder canvas))
(define not-redder-canvas (image.map (compose rgb.greener rgb.bluer) redder-canvas))
(image.show redder-canvas)
(image.show not-redder-canvas)
```

In case you were wondering, making the image greener and bluer does not invert the redder operation.

Explorations

Explorations are intended for students interested in further exploring the design aspects of these techniques. They also provide students who finish early with extra activities that may challenge them in different ways.

Exploration 1: Combining Transformations

While we only have a few basic transformations, there are, in some sense, an infinite number of ways to combine them. Try to find an interesting composition of basic transformations that someone might want to use as a filter.
