

Exam 1: Scheme Basics

Assigned: Friday, 8 February 2008

Due: Beginning of class, Wednesday, 13 February 2008

Preliminaries

Exam format

This is a take-home examination. You may use any time or times you deem appropriate to complete the exam, provided you return it to me by the due date.

There are 10 problems, each worth 10 points for a total of 100 points. Although each problem is worth the same amount, problems are not necessarily of equal difficulty.

Read the entire exam before you begin.

I expect that someone who has mastered the material and works at a moderate rate should have little trouble completing the exam in a reasonable amount of time. In particular, this exam is likely to take you about two to three hours, depending on how well you've learned the topics and how fast you work. You should not work more than four hours on this exam. Stop at four hours and write "There's more to life than CS" and you will earn at least 80 points on this exam.

I would also appreciate it if you would write down the amount of time each problem takes. Each person who does so will earn two points of extra credit. Since I worry about the amount of time my exams take, I will give two points of extra credit to the first two people who honestly report that they've spent at least three hours on the exam or completed the exam. (At that point, I may then change the exam.)

Academic honesty

This examination is open book, open notes, open mind, open computer, open Web. However, it is closed person. That means you should not talk to other people about the exam. Other than as restricted by that limitation, you should feel free to use all reasonable resources available to you. As always, you are expected to turn in your own work. If you find ideas in a book or on the Web, be sure to cite them appropriately.

Although you may use the Web for this exam, you may not post your answers to this examination on the Web. And, in case it's not clear, you may not ask others (in person, via email, via IM, by posting a please help message, or in any other way) to put answers on the Web.

Because different students may be taking the exam at different times, you are not permitted to discuss the exam with anyone until after I have returned it. If you must say something about the exam, you are allowed to say “This is among the hardest exams I have ever taken. If you don’t start it early, you will have no chance of finishing the exam.” You may also summarize these policies. You may not tell other students which problems you’ve finished. You may not tell other students how long you’ve spent on the exam.

You must include both of the following statements on the cover sheet of the examination.

1. I have neither received nor given inappropriate assistance on this examination.
2. I am not aware of any other students who have given or received inappropriate assistance on this examination.

Please sign and date each statement. Note that the statements must be true; if you are unable to sign either statement, please talk to me at your earliest convenience. You need not reveal the particulars of the dishonesty, simply that it happened. Note also that inappropriate assistance is assistance from (or to) anyone other than Professor Rebelsky (that’s me) or Professor Davis.

Presenting Your Work

You must present your exam to me in two forms: both physically and electronically. That is, you must write all of your answers using the computer, print them out, number the pages, put your name on the top of every page, and hand me the printed copy. You must also email me a copy of your exam. You should create the emailed version by copying the various parts of your exam and pasting them into an email message. In both cases, you should put your answers in the same order as the problems. Failure to name and number the printed pages will lead to a penalty of two points. Failure to turn in both versions may lead to a much worse penalty.

In many problems, I ask you to write code. Unless I specify otherwise in a problem, you should write working code and include examples that show that you’ve tested the code. Do not include images; I should be able to regenerate those.

Unless I explicitly ask you to document your procedures, you need not write introductory comments.

Just as you should be careful and precise when you write code and documentation, so should you be careful and precise when you write prose. Please check your spelling and grammar. Since I should be equally careful, the whole class will receive one point of extra credit for each error in spelling or grammar you identify on this exam. I will limit that form of extra credit to five points.

I will give partial credit for partially correct answers. I am best able to give such partial credit if you include a clear set of work that shows how you derived your answer. You ensure the best possible grade for yourself by clearly indicating what part of your answer is work and what part is your final answer.

Getting Help

I may not be available at the time you take the exam. If you feel that a question is badly worded or impossible to answer, note the problem you have observed and attempt to reword the question in such a way that it is answerable. If it's a reasonable hour (before 10 p.m. and after 8 a.m.), feel free to try to call me in the office (269-4410) or at home (236-7445).

I will also reserve time at the start of classes next week to discuss any general questions you have on the exam.

Problems

Recently, we have been exploring a fourth model of images, *spot lists*. As you may recall, a spot is a triplet that consists of column, row, and color. (We found different ways to represent these triplets with lists.) An image, then, is a list of spots. At the end of this exam, you will find the preliminary code to support spot lists. Note that this code includes a few new procedures.

The first set of new procedures involve a change to `image-scaled-render-spot!`. In place of that procedure, we now have two procedures, `image-render-spot-as-circle!` and `image-render-spot-as-square!`. We also have an *alias* that defines `image-scaled-render-spot!` as `image-render-spot-as-circle!`. To change how spots are scaled and rendered, you can change the alias to `image-render-spot-as-square!` or to one of the new scaled rendering procedures you will define in this exam.

In addition, two of the procedures extend rendering from single spots to lists of spots. The procedures (`image-render-spots!` *image spot-list*) (`image-scaled-render-spots!` *image spot-list factor*), use `image-render-spot!` and `image-scaled-render-spots!`, respectively, to render spots. Note that you do *not* need to understand how these procedures work; you just need to see that they do work.

Here is a list of spots to experiment with:

```
(define sample-spots
  (list
    (spot-new 0 0 "red")
    (spot-new 1 0 "orange")
    (spot-new 2 0 "yellow")
    (spot-new 3 0 "green")
    (spot-new 4 0 "blue")
    (spot-new 5 0 "indigo")
    (spot-new 6 0 "violet")

    (spot-new 0 1 "orange")
    (spot-new 1 1 "yellow")
    (spot-new 2 1 "green")
    (spot-new 3 1 "blue")
    (spot-new 4 1 "indigo")
    (spot-new 5 1 "violet")
    (spot-new 6 1 "red")
```

```

(spot-new 0 2 "yellow")
(spot-new 1 2 "green")
(spot-new 2 2 "blue")
(spot-new 3 2 "indigo")
(spot-new 4 2 "violet")
(spot-new 5 2 "red")
(spot-new 6 2 "orange")

(spot-new 0 3 "green")
(spot-new 1 3 "blue")
(spot-new 2 3 "indigo")
(spot-new 3 3 "violet")
(spot-new 4 3 "red")
(spot-new 5 3 "orange")
(spot-new 6 3 "yellow"))

```

You should try rendering the spots with both `image-render-spots!` and `image-scaled-render-spots!` to make sure you understand the behavior of the procedure and the pattern these spots make.

```

> (define canvas (image-new 200 200))
> (image-show canvas)
> (image-render-spots! canvas sample-spots)
> (define image-scaled-render-spot! image-render-spot-as-circle!)
> (image-scaled-render-spots! canvas sample-spots 10)
> (define image-scaled-render-spot! image-render-spot-as-square!)
> (image-scaled-render-spots! canvas sample-spots 10)

```

In some of the following discussion, we'll want to work with a single spot. Here's the default spot we will discuss.

```
(define sample-spot (spot-new 5 2 "red"))
```

Preparation

Before you begin, copy the definitions from the end of the exam to your definitions pane. Try the experiments described above.

Part A: Starting with Spots

1. Write a series of instructions that creates an interesting image by making a list of five spots (of your choice) and rendering them at different scales.
2. Write a procedure, `(spot-recolor spot newcolor)`, that creates a new spot at the same position as `spot`, but using the color `newcolor`.

```

> (spot-recolor sample-spot "blue")
(5 2 "blue")

```

Part B: From Spots to Drawings

When we have different representations of the same kind of value, we often write procedures that allow us to convert between representations. In this part of the examination, you will explore such conversion.

3. Write a procedure, (`spot->drawing spot`), that converts a spot to a drawing in which the spot is represented as a square. For example, if we start with a red spot at column 5 and row 2, your procedure should make a drawing by starting with the unit square, shifting it right 5.5 and down 2.5 and recoloring it red.

4. Write a procedure, (`scaled-spot->drawing spot hscale vscale`) that scales a spot to the appropriate drawing in which the spot is represented as an ellipse. For example, if we again start with the red spot at column 5 and row 2, and use an `hscale` of 10 and a `vscale` of 20, your procedure should rescale the unit circle to 10x20, recolor it red, and recenter it at (50,40).

Part C: Preventing Errors

As you may have noted, `image-scaled-render-spot!` can have some difficulties if the spot to be rendered does not fall within the bounds of the canvas. For example, suppose we tried to render our sample spot on a 50x50 canvas, scaled by a factor of 15.

```
> (define canvas (image-new 50 50))
canvas
> (image-show canvas)
(6)
> (image-render-spot-as-circle! canvas (spot-new 5 2 "red") 15)
Error: image-select-ellipse!: selection is outside of the bounds of the image
```

What's going wrong? As the error message suggests, we've called `image-select-ellipse!` (as you might expect), but the ellipse to be selected is outside the bounds of the image. What can we do when the spot will be rendered outside the bounds of the image (whether it's too far right, too far left, too far up, or too far down)? One possibility is to *wrap around*, so that spots that are a little too far to the right appear at the left, spots that are a little too far above the image appear at the bottom, and so on and so forth.

5. Write a procedure, (`image-render-spot-as-wrapped-circle! image spot factor`), that adds this wrap-around effect to `image-render-spot-as-circle!`.

This procedure should render each spot exactly once. Why do we note this restriction? Sometimes you will find that part of the scaled spot falls within the bounds of the image but that parts of the spot do not fall within those bounds. In these cases, ignore the extra bit of the scaled spot. For example, consider a spot at (2,1) scaled by 20 onto an image of size 45x45. The scaled top-left corner of the spot will be at (40,20), which will be on the image. However, since the diameter of the circle is 20, most of the circle will not appear, which is acceptable.

Part D: Different Ways to Render Spots

When we are dealing with purer representations of images, such as drawings and spot lists, it eventually becomes necessary to *render* the representation onto an image. You've already seen us do that with `image-render-spot!` and `image-render-spots!`. In this problem, we will consider alternative mechanisms for rendering spots.

6. Consider once again our sample red spot at column 2 and row 5. You will note that `(image-render-spot-as-circle! (spot-new 2 5 "red") 10)` renders the spot with a left edge of 20 and top of 50. Arguably, one might want the scaled spot *centered* at (20,50). Write a procedure, `image-render-spot-as-centered-circle!` that behaves much like `image-render-spot-as-circle!`, except that the circle is centered at its scaled position. That is, the column of the center should be `(* factor (spot-col spot))` and the row of the center should be at `(* factor (spot-row spot))`.

7. You may also have noted that when we scale spots at large scales, we get an awful lot of whitespace between spots. How can we alleviate this problem? We might make the radius of our spots a bit bigger than the `factor` parameter. Of course, this means that our spots will overlap a bit in places. Some call this a *fish-scale effect*.

Write a function, `image-fish-scaled-render-spot!` that it uses a fish-scale effect. In particular, you should render the spot as an ellipse, rather than a circle. The width of the ellipse should be 80% greater than `factor` and the height of the ellipse should be 20% greater than `factor`. Make sure to keep the ellipses centered at the scaled location.

8. An alternative to the fish-scale effect is to make bigger circles, but cut them off when they are about to overlap with the next circle. Write a procedure, `image-render-spot-as-squared-circle!`, based on `image-render-spot-as-circle!` that renders the spot as a circle of diameter 20% greater than `factor`, truncated to fall within a square whose side length is equal to `factor`. The following pictures illustrate the behavior of this new procedure.



The second image was created with commands like

```
> (define canvas (image-new 140 70))
> (image-show canvas)
> (define image-scaled-render-spot! image-render-spot-as-squared-circle!)
> (image-scaled-render-spots! canvas sample-spots 20)
```

Hint: Recall that we draw circles by selecting ellipses and then filling. Think of how we might truncate the circle before filling.

Part E: Generating Spots

We've explored how you might convert spots to another representation and how you might render them on a screen. However, we have yet to consider how to generate images that include a number of spots.

9. Sometimes, we might generate lists of spots from a single spot. For example, one might take a spot and make an "X" by listing that spot along with four more spots at positions diagonal to the spot.

Write a procedure, `(spot-x-seq spot)` that, given a spot, makes a list of five spots, including that spot and the four spots whose column and row differ from the first spot by exactly one.

```
> (spot-x-seq (spot-new 5 2 "red"))  
((4 1 "red") (6 1 "red") (5 2 "red") (4 3 "red") (6 3 "red"))
```

10. It might also be useful to do something like "create a line by filling in the spots from one point to another". Write a procedure, `(spot-5-seq color start-col start-row end-col end-row)`, that generates a list of five spots that begin at $(start-col, start-row)$, end at $(end-col, end-row)$, and are evenly spaced between the two places.

```
> (spot-5-seq "red" 0 0 100 0)  
((0 0 "red") (25 0 "red") (50 0 "red") (75 0 "red") (100 0 "red"))  
> (spot-5-seq "green" 0 0 0 60)  
((0 0 "green") (0 15 "green") (0 30 "green") (0 45 "green") (0 60 "green"))  
> (spot-5-seq "blue" 10 10 90 50)  
((10 10 "blue") (30 20 "blue") (50 30 "blue") (70 40 "blue") (90 50 "blue"))
```

Some questions and answers

Problem 1

Question: May I use more than 5 spots as long as they are in a list?

Answer: Yes.

Question: What do you mean by "render them at different scales"? Should I render each spot at a different scale?

Answer: You should use `image-scaled-render-spots!` to render the entire list of spots several times at different scales.

Problem 3

Question: For testing, is there a way to render a drawing on an image without creating a new image.

Answer: You can use `(drawing-render! drawing image)`.

Question: What will this procedure look like?

Answer: Something like

```
(define spot->drawing
  (lambda (spot)
    (drawing-hshift
     (drawing-vshift
      ...))))
```

Problem 10

Question: On problem 10, should I worry about what happens if the points in between the two endpoints have row or column values that are not integers?

Answer: DrFu ends up rendering these at the nearest pixel, which is fine.

Errata

Here you will find errors of spelling, grammar, and design that students have noted. Remember, each error found corresponds to a point of extra credit for everyone. I usually limit such extra credit to five points. However, if I make an astoundingly large number of errors, then I will provide more extra credit.

- We forgot the *scale* parameter in the textual description of *image-scaled-render-spots!*. [SR, 1 point]
- Missing subject in “scaled rendering procedures will define in this exam”. [AS, 1 point]
- In the introduction to Part C, wrote “spot to be rendered does not *all* within the canvas”. I meant “fall”. [SW and GF, 1 point]
- Scale factor misreported in problem 8. (The *diameter*, rather than the *radius* should be 20% greater than the scale factor.) [SW, 1 point]
- There were two definitions of *spot-nudge-up*. The second one should have been *spot-nudge-down*. Similarly, there were two definitions of *image-render-spot-as-circle!*, one of which should have been *image-render-spot-as-square!*. [GF and RZ, 1 point] (These errors had been corrected in the version of the exam distributed to one section, but not in the version distributed to the other. Both sections get the extra credit.)
- The *end-col* parameter was listed twice in Problem 10. One of these should be *end-row*. [SW, GF, and YY, 1 point]
- There was an error in the first definition of *image-render-spots!* [SW and GS, 1 point]
- Parameters were missing from example calls to *image-scaled-render-spots!* [AC]
- Problem 5 discussed the location of the center of the circle; however, *image-render-spot-as-circle!* is concerned with the top-left corner of the circle, not the center. [JN]
- In problem 2, “the same position at spot” should be “the same position *as* spot”. [AK]
- In problem 6, word “is” reduplicated in “the circle is is centered”. [AK]

A Library of Spot Procedures

```
(define spot-new
  (lambda (col row color)
    (list col row color)))
```

```

(define spot-col
  (lambda (spot)
    (car spot)))

(define spot-row
  (lambda (spot)
    (cadr spot)))

(define spot-color
  (lambda (spot)
    (caddr spot)))

(define spot-nudge-right
  (lambda (spot)
    (spot-new (+ (spot-col spot) 1) (spot-row spot) (spot-color spot))))

(define spot-nudge-left
  (lambda (spot)
    (spot-new (- (spot-col spot) 1) (spot-row spot) (spot-color spot))))

(define spot-nudge-down
  (lambda (spot)
    (spot-new (spot-col spot) (+ (spot-row spot) 1) (spot-color spot))))

(define spot-nudge-up
  (lambda (spot)
    (spot-new (spot-col spot) (- (spot-row spot) 1) (spot-color spot))))

;;; Procedure:
;;; image-render-spot!
;;; Parameters:
;;; image, an image
;;; spot, a spot
;;; Purpose:
;; Draw the spot on the image.
;;; Produces:
;;; [Nothing; Called for the side effect]
(define image-render-spot!
  (lambda (image spot)
    (context-set-fgcolor! (spot-color spot))
    (image-select-rectangle! image selection-replace
      (spot-col spot) (spot-row spot) 1 1)
    (image-fill! image)
    (image-select-nothing! image)))

;;; Procedure:
;;; image-render-spot-as-circle!
;;; Parameters:
;;; image, an image
;;; spot, a spot
;;; factor, a number
;;; Purpose:
;; Draw the spot on the image, as a circle whose diameter is factor.
;;; Produces:
;;; [Nothing; Called for the side effect]
;;; Preconditions:
;;; factor >= 1

```

```

;;; The position of the scaled spot is within the bounds of the image.
;;; Postconditions:
;;; The image now contains a rendering of the spot.
(define image-render-spot-as-circle!
  (lambda (image spot factor)
    (context-set-fgcolor! (spot-color spot))
    (image-select-ellipse! image selection-replace
      (* factor (spot-col spot))
      (* factor (spot-row spot))
      factor factor)
    (image-fill! image)
    (image-select-nothing! image)))

;;; Procedure:
;;; image-render-spot-as-square!
;;; Parameters:
;;; image, an image
;;; spot, a spot
;;; factor, a number
;;; Purpose:
;; Draw the spot on the image, scaled to a square whose edge length
;;; is factor.
;;; Produces:
;;; [Nothing; Called for the side effect]
;;; Preconditions:
;;; factor >= 1
;;; The position of the scaled spot is within the bounds of the image.
;;; Postconditions:
;;; The image now contains a rendering of the spot.
(define image-render-spot-as-square!
  (lambda (image spot factor)
    (context-set-fgcolor! (spot-color spot))
    (image-select-rectangle! image selection-replace
      (* factor (spot-col spot))
      (* factor (spot-row spot))
      factor factor)
    (image-fill! image)
    (image-select-nothing! image)))

(define image-scaled-render-spot! image-render-spot-as-circle!)

;;; Procedure:
;;; image-render-spots!
;;; Parameters:
;;; image, an image
;;; spots, a list of spots
;;; Purpose:
;; Draw all of the spots on the image.
;;; Produces:
;;; [Nothing; Called for the side effect]
(define image-render-spots!
  (lambda (image spots)
    (foreach! (lambda (spot) (image-render-spot! image spot)) spots)))

;;; Procedure:
;;; image-scaled-render-spots!
;;; Parameters:

```

```
;;; image, an image
;;; spots, a list of spots.
;;; factor, a number
;;; Purpose:
;; Draw all of the spots in the list on the image, scaled by factor.
;;; Produces:
;;; [Nothing; Called for the side effect]
;;; Preconditions:
;;; factor >= 1
;;; The position of the scaled spot is within the bounds of the image.
;;; Postconditions:
;;; The image now contains a rendering of each spot.
(define image-scaled-render-spots!
  (lambda (image spots scale)
    (foreach! (lambda (spot) (image-scaled-render-spot! image spot scale))
              spots)))
```

Copyright (c) 2007-8 Janet Davis, Matthew Kluber, and Samuel A. Rebelsky. (Selected materials copyright by John David Stone and Henry Walker and used by permission.)

This material is based upon work partially supported by the National Science Foundation under Grant No. CCLI-0633090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

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.