
Department of Computer Science, Australian National University
COMP1100 — Introduction to Programming and Algorithms
Semester 1, 2006

Assignment 1 — Image Manipulation

Due: 12 noon on Monday 3rd April, 2006

Outline

Your aim in this assignment is to write several computer image manipulation tools for **Portable PixMap** (ppm) images. The ppm format was discussed in lectures and a number of manipulation functions were developed.

The *assignments* page of the COMP1100 web site has a description of the representation of images on computers and the ppm format in particular.

A number of Haskell modules to represent pixels and images, and to convert to and from the ppm file format are provided in /dept/dcs/comp1100/public/assts/asst1/ and on the COMP1100 web site.

The main data types that will concern you are:

```
type Pixel = (Int,Int,Int)
type Row = [Pixel]
type Image = [Row]
```

There are also some functions, particularly in the `Pixel` module that will be useful. In the PPM module, `sampleImage :: Image` and `sampleContents :: String` may be useful when developing and testing your programs.

Conversion between the `Image` data structure above, and the ppm (text) format is achieved by functions provided in the PPM module. The `TestBed` module provides you with a means of testing your image manipulation functions.

Some sample ppm images are available in /dept/dcs/comp1100/public/assts/asst1Images/ and on the COMP1100 web site. If you wish to make your own images (make sure they're not too big) you can convert from other formats to ASCII ppm with a tool like Photoshop or the Gimp. *Eye Of Gnome* (eog) is a useful image display tool usually distributed with Linux, but there are others. Be sure to turn off interpolation to see true pixels when zooming in.

Required Tasks

You are required to design, develop and test several image manipulation tools. You should put each in a separate module with a sensible and informative name, along the lines of the sample modules `FlipH`, `FlipV` and `RedChannel`.

Desaturate

Write a function that converts an image to *greyscale*, that is, shades of grey. This is often called *desaturation*. For example:

```
desaturate :: Image -> Image
```

Hint: write a function to convert pixels to greyscale first. To convert a pixel to greyscale, each of its RGB values is changed to the average of the RGB values. (Round down to get an Int.) For example, (30,100,200) convert to (110,110,110).

As mentioned above, this should all be in a separate module with a sensible name such as `Desaturate`.

Double Scale

Write a function which doubles the size of an image (in both horizontal and vertical directions).

The simplest approach is to replace each pixel with 4 pixels.

For example the single (154,67,203) becomes:

```
(154,67,203) (154,67,203)
(154,67,203) (154,67,203)
```

A better approach is to introduce new pixels which interpolate between their immediate neighbours. For example, in the following diagram, the italicised pixels are the new ones:

```
(10,100,200) (30,150,225) (50,200,250)
(50,140,160)      ...      ...
(90,180,120)      ...      ...
```

Notice that with this approach, we don't get an exact doubling of the image size — an $n \times m$ image is scaled to size $(2n - 1) \times (2m - 1)$.

You may choose either means of scaling images. The first approach is much simpler, so make sure you can do that before attempting the (optional) interpolation technique.

Again, this should be in a separate module with a sensible name.

Free Choice Tasks

Successful completion of the *Required Tasks* above, to the satisfaction of your tutor and lecturer is sufficient to achieve a *Pass* grade on this assignment. For marks better than a *Pass*, you will need to develop some image manipulation tools of your own design.

There are two basic categories: *pixel-based* manipulations, where you modify each pixel in some manner (such as desaturation); and *image-based* manipulations, where the structure of the image is modified in some way (such as scaling). You should develop at least one of each kind. The difficulty of these image manipulation functions ranges from very easy to quite complex, so be aware of your own ability and expectations when choosing.

Pixel-based

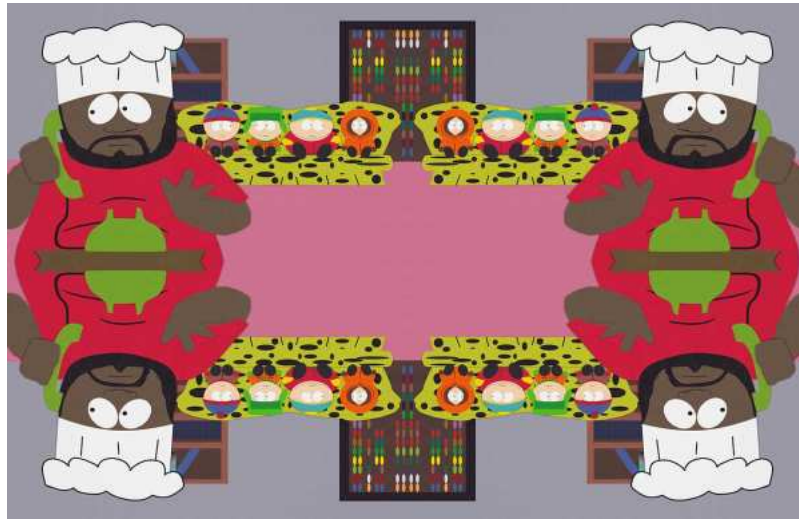
Here are a few suggestions for possible pixel-based manipulation functions. You are welcome — encouraged — to come up with your own ideas, but be sure to include an explanation of the technique and intentions in the comments in the corresponding Haskell modules you write. You may need to do some research to be sure you understand the technical definition of each manipulation.

- `invert :: Image -> Image`. Convert an image to its colour negative. Look at the information on colour representation on the web page accompanying this assignment. For example, cyan is the inverse of red and yellow is the inverse of blue.
- `darken :: Image -> Image`. Make an image twice as dark as it presently is.
- `threshold :: Int -> Image -> Image`. The brightness of a pixel is the average of the RGB values (see the `desaturate` function above). The first argument to `threshold` is a value between 0 and 255. All pixels at least as bright as that value are sent to white, and all pixels darker than that value are sent to black.
- `desaturate` again. The required exercise above gave us a complete desaturation to grey scale. It is more common to specify a degree of desaturation, say a value between 0 (for no effect) and 1 (for full desaturation).
- Increase the contrast of the image. You should be able to find a formula for changing contrast on the web or in the library.
- `posterize`. This is challenging and really belongs in the image-based manipulation group. First you will need to find out what it means. For thrill-seekers only.

Image-based

Here are a few suggestions for image-based manipulation functions. Once again, you are encouraged to come up with your own ideas, but give us a clear explanation.

- Rotate the image 180°. This is easy.
- Rotate the image 90°. This is not easy.
- Scale again. Rather than simply doubling the size of the image, provide the function with an argument to specify the scaling factor.
- Create a grid of four versions of the original image flipped along the edges of the original. There is an example below. Other variations are possible.



Submission

You should submit all the modules that you have written and wish to be included in your assessment, by the published deadline. Late submissions will be penalised 10% for each day (or part thereof) past the deadline.

Assignment submission will be electronic. Submit your assignment with the command:

```
submit comp1100 Asst1 <file names>
```

Only submit your own modules. You should not have modified any of the provided scripts (PPM.hs, Pixel.hs and so on).

Make sure you practice submitting your files well before the deadline. It is possible and perfectly acceptable to submit updated versions of your files. Our system will collect them all and your most recent submission will be the one assessed.

You should check that your submissions have been successful by clicking the **View Marks** button in **StReAMS** (<http://cs.anu.edu.au/streams/>). You should see a list of files together with the time and date of submission.

Assessment

This assignment forms 10% of your total assessment for COMP1100.

Successful completion of the *Required Tasks* to the satisfaction of your tutor and lecturer is sufficient to achieve a *Pass* grade on this assignment. For marks better than a *Pass*, you will need to develop some image manipulation tools of your own design, as explained in *Free Choice Tasks*.

There will be two components to the assessment. During week 7 practical classes you will present and demonstrate your solutions to your tutor and other members of the class. This will form 50% of your marks for this assignment. The second 50% will be based on your tutor inspecting your submitted modules. Marks will be allocated for programming style, comments, layout, technique and such.