

# COMP 40 Laboratory: Getting started with image compression

October 7, 2011

## Introduction

The new project is all about testing. If you go slowly and test methodically, you will have an easy time of it. If you write a big pile of code before testing, debugging it will be NO FUN. This lab shows you *exactly* how to get started on testing. There is nothing to submit, but if you follow the instructions, you will finish most of your compressor quickly and easily. In next week's lab, we'll talk about the Bitpack module.

## Plan of the lab

1. Design the `ppmdiff` program, specified below, which will allow you to compare two images programmatically. The `ppmdiff` program will help you test your compressor and decompressor.
2. Before building anything, *check your design with the course staff*.
3. Build `ppmdiff`.
4. Build a trivial image transformer which goes from the top of Figure 1 (page 4) down to the third box and back up to the top again.
  - (a) Read an image in PPM format
  - (b) Trim to an even number of columns and rows
  - (c) Convert from scaled integers to floating-point numbers
  - (d) Convert from floating-point numbers to scaled integers
  - (e) Write the image in PPM format
5. Use `ppmdiff` to make sure your image transformer works properly. (You can also confirm by using `display` on the images.)
6. Built a slightly more elaborate version of the transformer from step 4. The more elaborate transformer should add one more pair of transformations: conversion from RGB color space to  $Y/P_R/P_B$  color space and back again.
7. Test the more elaborate transformer.

After completing these steps, you'll be well positioned to keep extending your image transformer until you eventually have a complete compressor and decompressor.

## Specification of `ppmdiff`

Program `ppmdiff` takes two arguments on the command line. Each argument is the name of a PPM file. Optionally, one or the other argument (but not both) may be the C string "-", which stands for standard input. Here's what `ppmdiff` does:

- Both files are read into images  $I$  and  $I'$ . The width and height of  $I$  and  $I'$  should differ by at most 1; if the difference is larger, `ppmdiff` should print an error message to standard error and should print the number 1.0 to standard output.
- Assuming  $w$  and  $h$  represent the *smaller* of the two widths and heights, compute

$$E = \sqrt{\frac{\sum_{0 \leq i < w} \sum_{0 \leq j < h} (R_{ij} - R'_{ij})^2 + (G_{ij} - G'_{ij})^2 + (B_{ij} - B'_{ij})^2}{3 \times w \times h}}$$

where, for example,  $R_{ij}$  is the red pixel located at coordinate  $(i, j)$  of image  $I$ . The value  $E$  is the *root mean square difference* of the pixel values in the two images.

- Print  $E$  to standard output with four digits after the decimal point.

## Reminders

- Get an end-to-end solution working as quickly as possible, then improve it.
- Compile insanely often. If you use Emacs, learn how to use the commands

```
M-x compile
C-x ‘
```

which will take to straight to the place where errors occur.

If you use Vim, learn to use the commands `:make` and `:cn`.

For either editor, your “compile command” should be `sh compile`, not `make` or `make -k`.

- Every time you extend your transformer, run it and compare the results using `ppmdiff`.
- Every time you get a good answer with `ppmdiff`, run your code again with `valgrind`.

## What to expect from `ppmdiff`

Two similar but not identical images have a difference of around 16%:

```
: nr@labrador ; ppmdiff a.ppm b.ppm
0.1656
```

Pictures that are not at all similar have a larger difference:

```
: nr@labrador ; ppmdiff a.ppm c.ppm
0.2628
```

And just taking a single image, compressing with JPEG, and decompressing it, can produce errors from 0.1% to 1.5% (sometimes as high as 2.5%):

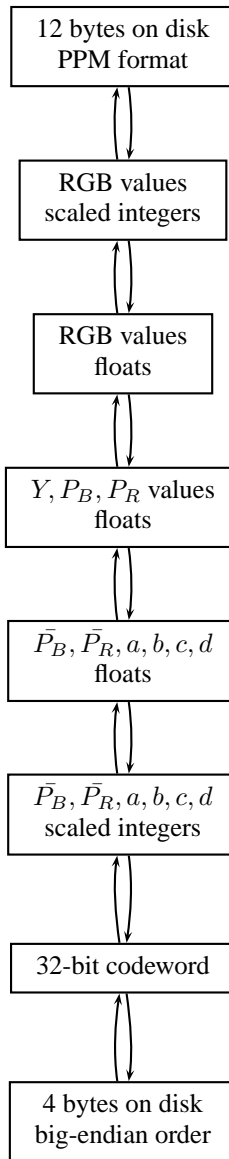
```
: nr@labrador ; cjpeg cc.ppm | djpeg | ppmdiff cc.ppm -
0.0013
: nr@labrador ; cjpeg gullfoss.ppm | djpeg | ppmdiff gullfoss.ppm -
0.0165
```

The numbers above are artificially low, because the original images have *already* been compressed with JPEG, so what we’re seeing is the *additional* error introduced on a later run. If we use JPEG to compress and decompress a lossless image like a PNG, we see a larger error of around 2.5%:

```
: nr@labrador ; cjpeg qc.ppm | djpeg | ppmdiff qc.ppm -
0.0255
```

On these kinds of images, the compressor you build should be competitive in quality with JPEG:

```
: nr@labrador ; 40image -c qc.ppm | 40image -d | ppmdiff qc.ppm -  
0.0266  
: nr@labrador ; 40image -c cc.ppm | 40image -d | ppmdiff cc.ppm -  
0.0223  
: nr@labrador ; 40image -c gullfoss.ppm | 40image -d | ppmdiff gullfoss.ppm -  
0.0225
```



Compression goes from the top representation to the bottom representation. Decompression goes from the bottom representation back to the top representation.

Figure 1: Representations of a  $2 \times 2$  block