COMP 40 Laboratory: Getting started with image compression

Introduction

The new arith project, which we have just released, requires a serious focus on 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. Every year the students who take this advice do much better and spend much less time than those who don't. 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.

Plan of the lab

In the arith project, you will design a compressor for color images. The compressor works in stages, transforming the input image by computing summary numbers (such as the average brightness), and packing approximations to those values into bitfields in a code word. Because the code word has fewer bits than the input pixel values it represents, the resulting image file is compressed. You also write an image decompressor that reconstruct the image from the compressed form. In this lab you will do two things:

1. Write a diff program that can compare two images to see get a quantitative estimate of how similar they are. Later, you will use this in your testing to see how good your compressor/decompressor is at preserving the input image.

2. You will start on arranging your compressor/decompressors to allow for you test it’s individual compression stages separately. This is important! Students who follow this strategy will typically be more successful!

We know that the arith assignment has just been released. Don’t worry if it’s too early to make progress here in lab on the second item above. Do be sure to keep these instructions and follow them carefully later when you build and test your compressor!

Here’s what to do for today’s 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. If you have questions about ppmdiff, check with the course staff before writing code.

2. Code and test ppmdiff here in the lab.

3. With the rest of your lab time, start building 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

Although your full image transformer will write and read compressed files to/from disk, we’ll leave that step for later. For testing purposes, you’ll have your program compress “part of the way,” and then immediately reverse the transformation. For this first step, you’re only going as far as encoding into floating point numbers, so you immediately enter your decompression path and:
4. Use `ppmdiff` to make sure your image transformer works properly. (You can also confirm by using `display` on the images.)

5. Built a slightly more elaborate version of the transformer from step 3. 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.

6. 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. It writes to standard output a single number which is a measure of the difference between two input images. 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**

- After reviewing all of the lab instructions, build and test your `ppmdiff` program
- Get a preliminary, limited function end-to-end (compress/decompress) `ppmtrans` solution working as quickly as possible. then improve it incrementally.
- Compile insanely often. If you use `Emacs`, learn how to use the commands
  
  `M-x compile`
  `C-x '`

  which will take you 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%:

```bash
$ ppmdiff a.ppm b.ppm
0.1656
```

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

```bash
$ 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%):

```bash
$ cjpeg cc.ppm | djpeg | ppmdiff cc.ppm -
0.0013
$ 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%:

```bash
$ 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:

```bash
$ 40image -c qc.ppm | 40image -d | ppmdiff qc.ppm -
0.0266
$ 40image -c cc.ppm | 40image -d | ppmdiff cc.ppm -
0.0223
$ 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