Project 4: Map-Reduce Medical and Scientific Image Enhancement
15-440 Spring 2012

Assigned: Thursday, 5 April
Due: Thursday, 19 April

Overview

This project asks you to study the supplied code for enhancing high resolution medical and scientific images and develop from it a Map-Reduce distributed task that performs the same enhancement – but has the capability to process, in a timely way, images many times the size. The image processing technique uses a Laplacian Pyramid to find and enhance edges in frequency space, and tunes the greyscale to enhance contrast. One of the important challenges of this project is identifying the opportunities for parallelism within the processing of a single large image, without generating offsetting inefficiency due to overhead. For your image processing enjoyment, we've included mammograms, for which this technique has been previously studied and has dramatic results, as well as lunar images, for which we've observed that it is also very effective.

Logistics

You should work with a partner on this project. A complete sequential implementation of the Laplacian Pyramid method is provided in the Labs section of the course website. Your first task, and perhaps most important, is to study this code to identify opportunities for parallelism exploitable by Map-Reduce. Feel free to re-use any of the code provided. When you are finished, submit a single zip archive to:

`/afs/andrew/course/15/440-s12/handin/lab4/yourandrewid-partnerandrewid`.

where `yourandrewid` is the Andrew ID of one of the partners in the group. Mention the names and IDs of both partners in a comment at the top of every file that you write. Be sure to include all files necessary to compile and run the project (including a Makefile). The GHC 5xxx machines (ghc01.ghc.andrew.cmu.edu - ghc81.ghc.andrew.cmu.edu) are running version 1.01 of Hadoop. Your project must run on the Gates Hadoop cluster, but you should certainly feel free to download a copy of the same version of Hadoop and Java to your own personal machine for testing on small image files.

Detailed Description of the Laplacian Pyramid method

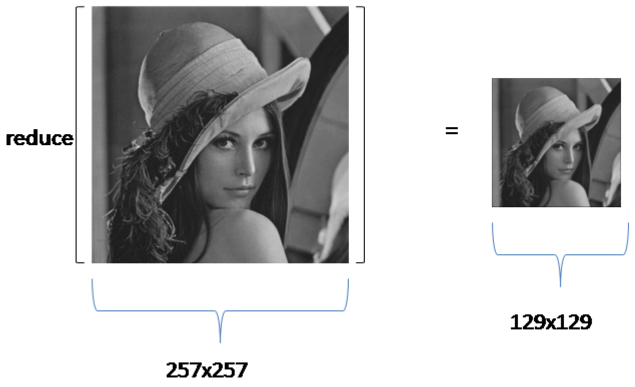
The Laplacian Pyramid was originally described as a technique to encode images in a compact way [1]. Here we will describe an algorithm that uses Laplacian Pyramids to enhance the contrast of greyscale

[1]The original paper by Burt is recommended reading. A link is provided in the reference section.

images[2]. For much of this section, we will view an image as a 2D matrix of pixels, where (for greyscale images) each pixel is represented by an integer from 0 - 255, with 0 indicating that pixel is black and 255 indicating that pixel is white. Values in between indicate that pixel is some shade of grey.

Building the Gaussian Pyramid with `reduce`

We define a function `reduce()` which takes an image and downsamples the image to approximately half its original resolution. In terms of matrices, this takes a $(2M+1)$ by $(2N+1)$ matrix and returns a matrix of size $(M+1)$ by $(N+1)$, such that the matrices correspond to similar looking images. Here is an example:



Each pixel in the smaller image is composed of a weighted average of a block of 5x5 pixels in the larger image. Specifically, for i, j, $0 \leq i < M+1$ and $0 \leq j < N+1$ we have:

$$image_{small} = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) * image_{large}(2i+m, 2j+n)$$

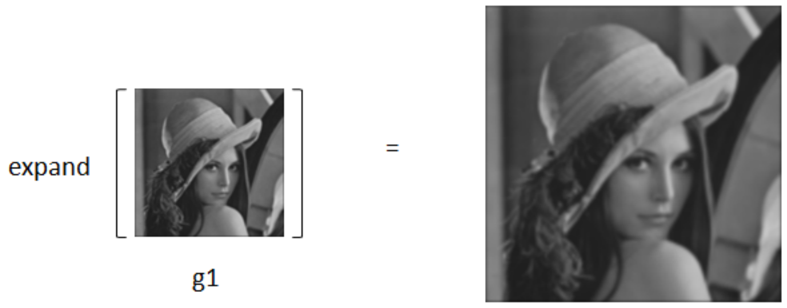
where $w(m,n) = \hat{w}(m) * \hat{w}(n)$ and $\hat{w}(0) = 0.05$, $\hat{w}(1) = 0.25$, $\hat{w}(2) = 0.40$, $\hat{w}(3) = 0.25$, and $\hat{w}(4) = 0.05$. This function is implemented in the handout code as PGMTransform.reduce().

`reduce()` is used repeatedly to create a set of images, each roughly half the size of the next. This is also known as the Gaussian Pyramid (so called because of how the images can be stacked on top of one another to sort of form a pyramid). We denote the first image (the original image) as g_0, and the next as g_1, etc and in general $g_i = \mathtt{reduce}(g_{i-1})$ Note that the Gaussian Pyramid is built bottom-up. Here is a 5 level Gaussian Pyramid for this image:

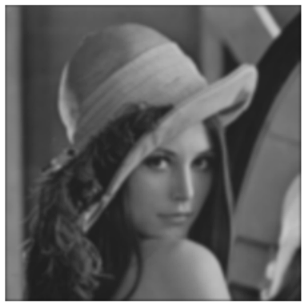
[2]A paper by Dippel et al, listed in the reference section, is the basis for this project and the source for the "Woman with hat" images in this handout.

Building the Laplacian Pyramid with expand and subtract

We define a function expand() which takes an image and upsamples the image to approximately twice its original resolution. In terms of matrices, this takes a $(M+1)$ by $(N+1)$ matrix and returns a matrix of size $(2M+1)$ by $2(N+1)$, such that the matrices correspond to similar looking images. Here is an example:



g1,1 (not g0)



expand(reduce(g0))

g0

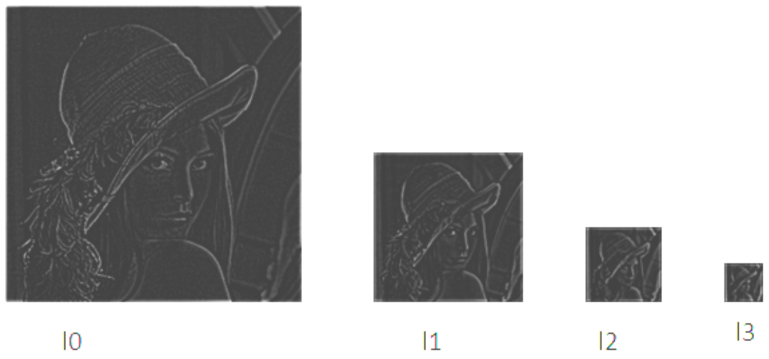
Note that expand() is not the inverse of `reduce()`, because information is lost in the reduction step so the original cannot be perfectly reformed. Each pixel in the larger image is created by interpolating new values from the smaller image. Specifically, for i, j, $0 \leq i < 2M+1$ and $0 \leq j < 2N+1$ we have:

$$image_{large} = 4 \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) * image_{small}\left(\frac{i-m}{2}, \frac{j-n}{2}\right)$$

Only terms for which $(i - m/2$ and $(j-n)/2$ are integers are included in this sum. $w(m,n)$ is the same function as in expand(). This function is implemented in the handout code as PGMTransform.expand().

The Laplacian Pyramid is built by expanding and subtracting successive levels of the Gaussian Pyramid. Subtraction is done pixel-wise, and is the same subtraction function defined on matrices, except that negative values are padded to zero. That is, $l_0 = subtract(g_0, expand(g_1))$, and in general $l_i = subtract(g_i, expand(g_{i+1}))$. The only exception is the last level, for which there is no g_{n+1}. We define $l_n = g_n$ for this reason.

Here is the Laplacian Pyramid taken from the 5 level Gaussian Pyramid from before (l_4 not shown).



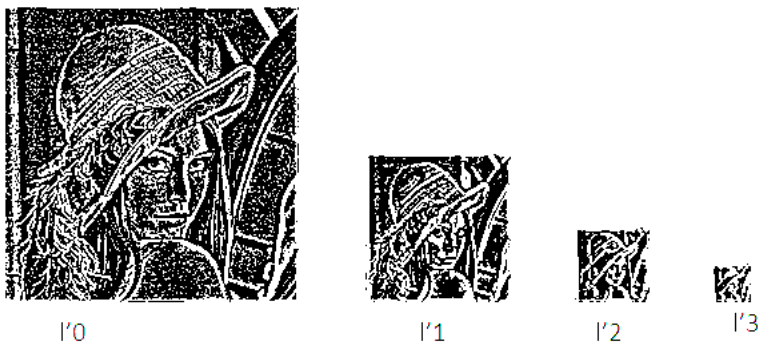
Note that the Laplacian Pyramid has all of the essential features of the image without any background detail.

Applying the Low-pass Filter

In this step we enhance the essential features of the image by brightening the dark grey pixels in each image of the Laplacian Pyramid, while leaving the brighter pixels alone. This is accomplished by applying a low-pass filter to each level (image) of the Laplacian Pyramid to obtain our modified Laplacian Pyramid. Our particular low-pass filter is given by the function q:

$$q(x) = \begin{cases} Gx\left(\frac{1-x}{30}\right)^p & : x < 40 \\ x & : x \geq 40 \end{cases}$$

where G and p are parameters that are different for each level. The values are experimentally chosen to fit the data set well. This function is implemented in the handout code as PGMTransform.quantize(). You can see the parameters for each level in contrast/PGMConstrast in the handout code. You need not change those parameters for your implementation. Here is the modified Laplacian Pyramid:

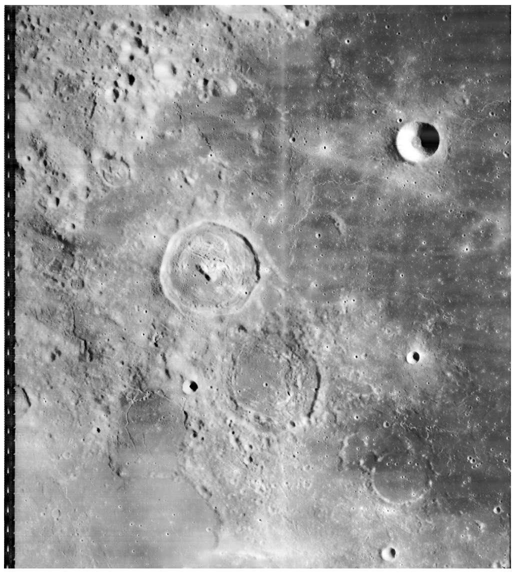
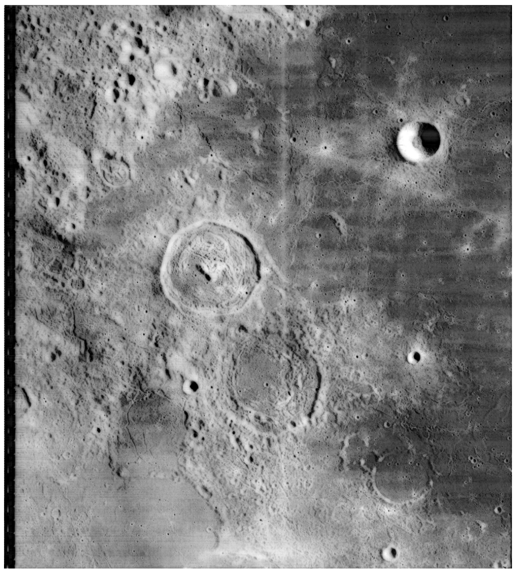


You can see that the essential features are even more noticeable now. Note that the quantization step is NOT performed on l_n - not shown. The quantization step is only applied to images l_0 through l_{n-1}. We define $l'_n = l_n$.

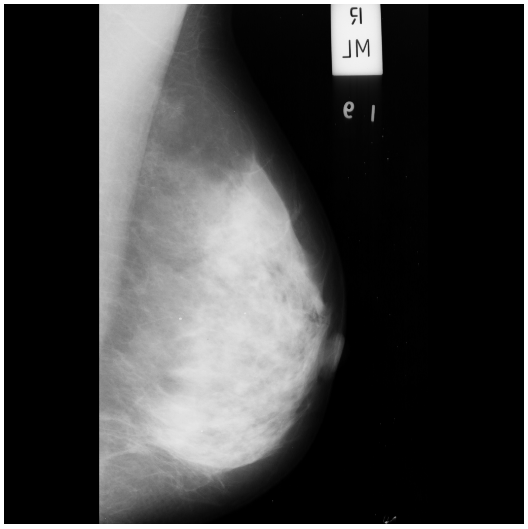
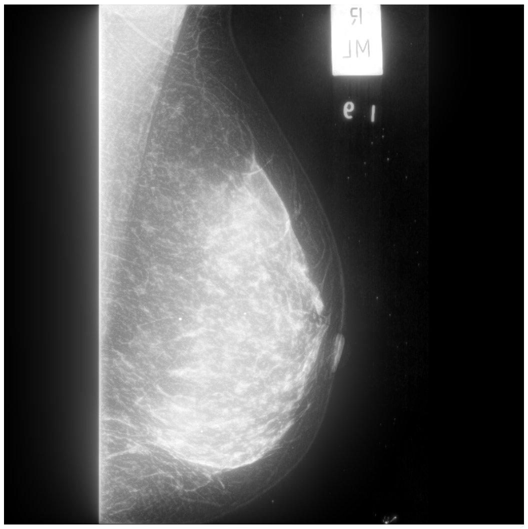
Reconstruction of the Contrast Enhanced Image

A Gaussian Pyramid is constructed from the modified Laplacian Pyramid, and the bottom level of the new Gaussian Pyramid is given back as the contrast enhanced image. The new Gaussian Pyramid is built top-down. The top level g'_n is the same as l'_n. For each level i, from $n-1$ to 0, $g'_i = add(l'_i, expand(g'_{i+1}))$. g'_0 is the contrast enhanced image.

Our image we've used thus far to illustrate the process doesn't contrast enhance nicely, so here's a picture of the moon, which does. Pay particular attention to the ridges between the craters and other small details. (Actually, this image was inverted, contrast enhanced, and then inverted again, because the highlights were shadows in the original image).

Perhaps the most impressive images are the mammograms, for which this technique was developed and tuned. For example, consider these images:

In any case, please be aware that the results will seem much more impressive when you view the full-size images and are able to see the fine detail — check out the data sets, described below.

Data sets

There are four data sets available for you to use. They all exist in the HDFS space, under `/data`. The handout data set is good for testing your parallel implementation. The mias-database data set is a collection of mammograms from the Mammographic Image Analysis Society, which fit in memory, which might enable you to start from a simpler place. The large data set consists of several images which are 64MB+ in size. The huge data set is a good test of how well your solution works on extremely large images. To earn full credit your project must work on the mias-database, large and huge data sets. Only one data set per run, please.

Implementation Details

You will perform contrast enhancement on each image with the same parameters as the handout code. Each pyramid will have 6 levels to it. You may assume that the image has the right dimensions to construct a 6-level pyramid without need for padding or cropping the image. You may not assume that the image will have equal dimensions.

Your implementation should do as many steps in parallel as possible, as long as it makes sense to do so. Be sure to not re-do expensive computations (if you are wondering what 'expensive' is, consider larger images). Hint: If you split images, many nodes can work on the low-pass filter step in parallel. You should consider whether or not other steps are worth doing in parallel. You should also consider how to best split images or otherwise divide the work to be done. Recall that the pixels in the .pgm file are stored in row-major order. Your implementation need only be concerned with working on portable greymap (.pgm) images.

Additionally, the user of your project must be able to set the output directory and desired dataset via command line arguments.

Prizes will be awarded to the three fastest implementations, as judged by their performance on the huge data set.

Using The Gates Hadoop Cluster

You will want to work through the word count example in the Yahoo! tutorial (see references section) to familiarize yourself with Hadoop.

Once you have done so, you may access the Gates Hadoop cluster by logging onto ghcxx.ghc.andrew.cmu.edu (xx ranging from 1 to 81). Add /usr/local/hadoop/bin to your $PATH environment variable and set $JAVA_HOME to /usr/lib/jvm/jre-sun. If you're using csh, then the commands are:

```
setenv JAVA_HOME /usr/lib/jvm/jre-sun
setenv PATH $PATH:/usr/local/hadoop/bin
```

You may then run hadoop commands as normal. The 440 staff wishes that you become familiar with these two commands in particular: `hadoop job -list`, which lists currently running jobs, and `hadoop job -kill <jobid>`, which kills a running job.

Additionally, you can monitor the Map-Reduce engine here:

`http://ghc81.ghc.andrew.cmu.edu:50030/jobtracker.jsp`

And the scheduler here:

`http://ghc81.ghc.andrew.cmu.edu:50030/scheduler`

And the HDFS file system here:

`http://ghc81.ghc.andrew.cmu.edu:50070/dfshealth.jsp`

The 440 staff is well aware that it is a popular strategy to wait until the last minute to start projects. This will have the effect of overloading the Gates Hadoop cluster until it slows to a crawl. Please take care to kill your own long running jobs that aren't doing anything useful. If you notice that the Hadoop cluster is prohibitively slow, then send an email to the 440 staff list and we will remedy the problem by mercilessly slaughtering all 440 related jobs that have run for too long.

You should be aware that the cluster has been configured to use a priority scheduler that allocates resources preferentially to those who have recently consumed the least and that you should not submit more than one job to the cluster at a time, and that the system will not dispatch more than three jobs at a time.

References

http://developer.yahoo.com/hadoop/tutorial/index.html. The Yahoo Hadoop Developer Tutorial.

The Laplacian Pyramid as a Compact Image Code. The original paper which describes how to generate a Laplacian Pyramid and why they are useful. See paper1.pdf in the Labs section.

Multiscale Contrast Enhancement for Radiographies: Laplacian Pyramid Versus Fast Wavelet Transform. Describes contrast enhancement with Laplacian Pyramids as applied to various radiographies. See paper2.pdf in the Labs section.