Elec4622 Laboratory Project 2, T2 2024

David Taubman

July 5, 2024

1 Introduction

This is the second of three mini-projects to be demonstrated and assessed within the regular scheduled laboratory sessions. This project is due in Week 8. The project is worth a nominal 10 marks, out of the 30 marks available for the laboratory component of the course. However, some suggested activities can attract substantial bonus marks.

This project is concerned with implementing a Laplacian pyramid and using it to process images in the pyramid domain. Here is a brief summary of the relevant theory and concepts.

1.1 Creating the Gaussian Pyramid

Starting with the image $x[\mathbf{n}] = x_0[\mathbf{n}]$ a stack of D reduced resolution images $x_1[\mathbf{n}], \ldots, x_D[\mathbf{n}]$ may be created by low-pass filtering and downsampling, as follows:

$$\begin{aligned} x_d[\mathbf{n}] &= (x_{d-1} * h)[2\mathbf{n}] \\ &= \left\langle \mathbf{x}_{d-1}, \tilde{h}_{2\mathbf{n}} \right\rangle \\ &= \sum_{\mathbf{k}} x_{d-1}[\mathbf{k}]\, \tilde{h}[\mathbf{k} - 2\mathbf{n}] \\ &= \sum_{\mathbf{k}} x_{d-1}[2\mathbf{n} + \mathbf{k}]\, \tilde{h}[\mathbf{k}] \end{aligned}$$

where $h[\mathbf{n}]$ is the PSF of a suitable low-pass filter. The most natural implementation strategy is the output-driven inner-product approach (moving average window) where $\tilde{h}[\mathbf{k}]$ are the weights, $2\mathbf{n}$ corresponds to the placement of the "window" and $\mathbf{k}$ runs over the support of this weighting window, $\tilde{h}$.

To minimise aliasing, we would like to have

$$\hat{h}(\boldsymbol{\omega}) \approx 0 \text{ whenever } |\omega_1| \geq \frac{\pi}{2} \text{ or } |\omega_2| \geq \frac{\pi}{2}$$

We could use Gaussian filters, as in Project 1, but in this project we will use Windowed sincs. You will can use any smooth window function, although the Hanning window is recommended for simplicity. The only parameter of interest is the region of support of the separable windowed sinc $h[\mathbf{n}] = h_1[n_1] \cdot h_1[n_2]$.

1.2 Creating the Laplacian Pyramid

At each level $0 \leq d < D$ of the Gaussian pyramid, a Laplacian detail image can be formed according to

$$\mathbf{y}_d = \mathbf{x}_d - \sum_{\mathbf{k}} \mathbf{x}_{d+1}[\mathbf{k}]\, \mathbf{g}_{2\mathbf{k}}$$

where $g[\mathbf{n}]$ is a low-pass interpolation filter and the second term on the right hand side of the above equation represents interpolation of the reduced resolution image $x_{d+1}[\mathbf{n}]$ up to the resolution of $x_d[\mathbf{n}]$. Note that each sample $x_{d+1}[\mathbf{k}]$ of the reduced resolution image acts as an excitation source for the interpolation filter, where the location of the excitation source (in the interpolated domain) is $2\mathbf{k}$. The above equation can be written equivalently as

$$y_d[\mathbf{n}] = x_d[\mathbf{n}] - \sum_{\mathbf{k}} x_{d+1}[\mathbf{k}]\, g[\mathbf{n} - 2\mathbf{k}]$$

The most natural implementation strategy for this operation is the input-driven one. Start by copying the contents of x_d to the memory buffer that will hold y_d; then walk through each location $\mathbf{k}$ within x_{d+1}, scaling the elements of g by the value of x_{d+1} and subtracting the resulting values from the appropriate locations within the memory buffer that holds y_d.

As above, the PSF $g[\mathbf{n}]$ may be a Gaussian or another suitable low-pass filter, whose bandwidth should ideally be confined to the box $\boldsymbol{\omega} \in \left(\frac{-\pi}{2}, \frac{\pi}{2}\right)^2$. For this project, we will use windowed sincs. It is your responsibility to choose suitable stretching factors for the sinc, and also to make sure that your upsampling and downsampling operations all have a DC gain of 1.

1.3 Inverting the Laplacian transform

Starting with $y_D = x_D$ and the detail images y_0 through y_{D-1}, it is a simple matter to recover $x[\mathbf{n}] = x_0[\mathbf{n}]$ through a succession of interpolation and addition operations. That is,

$$x_d[\mathbf{n}] = y_d[\mathbf{n}] + \sum_{\mathbf{k}} x_{d+1}[\mathbf{k}]\, g[\mathbf{n} - 2\mathbf{k}]$$

1.4 Stitching in the Laplacian domain

In this project, we will explore the use of the Laplacian transform for image stitching. To make things simple, we will create two source images $x^A[\mathbf{n}]$ and $x^B[\mathbf{n}]$, to be stitched right down the middle. That is, direct image domain stitching should produce

$$x^S[\mathbf{n}] = \begin{cases} x^A[\mathbf{n}] & 0 \leq n_1 < \lfloor \frac{N_1}{2} \rfloor \\ x^B[\mathbf{n}] & \lfloor \frac{N_1}{2} \rfloor \leq n_1 < N_1 \end{cases}$$

where N_1 is the image width and both images have identical dimensions.

To perform the stitching in the Laplacian pyramid domain, you should first convert x^A and x^B into their Laplacian sub-images, $y_0^A, \ldots, y_D^A$ and $y_0^B, \ldots, y_D^B$, perform the stitching separately in each level of the Laplacian pyramid, and then synthesise the stitched Laplacian sub-images $y_0^S, \ldots, y_D^S$ back to a final stitched image x^S.

2 How to go about the tasks

For demonstration purposes, you should create a separate project for each task, within a single workspace. You can do this by using the "File → New → Project" option in Visual C++, replacing the "Create new solution" option with "Add to solution".

Be sure to arrange for all your executable programs to be placed in a single location (e.g., `c:\elec4642\bin`), which is referenced from the `path` environment variable. Also, please place all the images you are working with into a single directory (e.g., `c:\elec4642\data`). That way, it will be much easier to demonstrate your work in a time effective manner – it will also save you personally a lot of time. In any event, **you might not be marked for your work unless it is organized in this way**.

You are expected to produce **hand-drawn sketches** illustrating key features of your design for each task. This is not really any additional work, since it would normally be part of the process of coming up with a design and getting it working. As explained already in laboratories, being able to draw what a diagram of what your program is doing is an essential first step to implementing a solution and a key element in the debugging process. **Do not discard these sketches and diagrams, since lab demonstrators will expect to see them when marking your work, and you are also required to upload them to Moodle as part of the electronic submission process.**

To verify your work, it may be helpful to collect sequences of command-line statements together into a script that you can run quickly. You can do this with a Windows batch file (i.e., any file with the ".bat" suffix), which can then be executed like any other program from the command-line. This will greatly facilitate the marking for demonstrators, who might insist upon it.

3 Tasks

Task 1: (4 marks) Write a program that accepts an input BMP image $x\,[\mathbf{n}]$ and converts it to a Gaussian pyramid with D levels. Your program should accept D as a command-line argument, and it should write out all $D+1$ sub-images $x_0, \ldots, x_D$ as a single BMP image, with x_0 at the top, followed by x_1 (each row padded with $N_1/2$ zeroes on the right) and so on until x_D appears at the bottom of the output image. The resulting output image should look like an upside down pyramid, with all sub-images lining up with the left hand border.

Your program should also accept an integer parameter H that represents the half-length of the windowed sinc $h\,[n]$ that is separably extended to $h\,[\mathbf{n}] = h_1[n_1] \cdot h_2[n_2]$ such that $h_1\,[n] = 0$ for $|n| > H$. That is, your 2D windowed sinc filters or interpolation kernels should all have region of support $[-H, H]^2$. Your program should be able to operate successfully with value of H that are as small as 0 and as large as 20.

For simplicity, you are recommended to use floating point arithmetic internally within your program. This means that your sub-images $x_d\,[\mathbf{n}]$ would be stored as floats, but you will need to take care when converting the results back to 8-bit unsigned integers and writing out your BMP image.

Make sure your program works for colour images.

Task 2: (4 marks) Modify the program from Task 1 in such a way that the output image holds the Laplacian pyramid sub-images $y_0, \ldots, y_D$. In this case, for display purposes only, add 128 to all the detail images $y_0, \ldots, y_{D-1}$, so that positive and negative values are readily visible as offsets relative to the mid-level grey signal.

As in Task 1 you will need to be careful when converting floats to 8-bit integers for display, and you will also have to pay close attention to clipping the values to the limits of 0 to 255 – why is this much more important for this task than for Task 1?

Make sure your application works for colour images, producing colour Laplacian sub-images.

Task 3: (2 marks) Write a program that accepts the Laplacian output image produced by Task 2 and reconstructs the original input image, writing it out as a BMP image.

You may find it convenient to supply the height of the original image on the command line so that you don't have to figure this out from the height of the pyramid image.

You should verify that your program works by comparing the original input image with the one that you reconstruct using the Media Interface `"image_arith"` module – e.g.,

mi_pipe2 -in1 orig.bmp -in2 recon.bmp :: image_arith -add 1 -1

To explore discrepancies more readily, try doing some brightening of the output image as follows:

mi_pipe2 -in1 orig.bmp -in2 recon.bmp :: image_arith -add 1 -1 :: cc_interp -br 10

Make sure you can explain any discrepancies!

Task 4 (optional): (up to 1 bonus mark) Combine tasks 1, 2 and 3 into a single program that reads an input image, creates the Laplacian domain representation internally, then reconstructs the image and writes it out, doing all internal processing in floating point (or, if you like, fixed-point) arithmetic. How does your output image compare with the original input image in this case? To verify that you are doing this exercise correctly, the lab demonstrator may ask you to **multiply the samples of one of the Laplacian detail images by a fixed factor** prior to reconstruction.

Task 5 (optional): (up to 1 bonus mark) Write a program that accepts two BMP input images, x^A and x^B, stitching them together **in the image domain** and writing the stitched image out as another BMP file.

Your program can assume that the images have identical dimensions and it is sufficient to place a vertical stitching boundary at a fixed location in the middle of the two images.

Task 6 (optional): (up to 2 bonus marks) Modify your program from task 4 so that it accepts two input images of the same size, stitches them **in the Laplacian Pyramid domain**, and then reconstructs and writes the output BMP image. To test the behaviour of your program, prepare ligher and darker versions of a single input image as x^A and x^B and stitch them together. You can prepare such images by using the Media Interface `"cc_interp"` module with the `"-br"` argument, as illustrated above.

Task 7 (optional): (up to 1 bonus mark) Prepare an original image and a blurred version of the original image and stitch them together using the programs from Task 5 and Task 6, comparing the results visually.

- To blur an image, you can use the Gaussian filtering program that you implemented for Task-1 of Project-1, but this task does not impose specific constraints on how you obtain the blurred version of an image.

4 Assessment

You should not rely upon implementing this project within the scheduled laboratory sessions. Instead, you must be prepared to demonstrate and explain your work in the Week 8 laboratory. You should make sure each task can be run simply from the command-line. Spend some effort organizing your patterns and test images ahead of time, perhaps using separate sub-directories. I suggest creating small batch files (files ending in ".bat" that contain commands you would normally type on the command line) that display the test image and output using "mi_viewer". Doing this will save you and the demonstrator a lot of time, both before and during the lab, since it is easy to compare different tasks using such batch files without having to remember all the conditions you used.

Note carefully: Lab demonstrators will expect to see the hand-drawn sketches you have produced as a critical part of the design process, showing how you have organized your image data, how you have computed dimensions, extended boundaries, and other such things.

4.1 Team work, plagiarism and copying

You may feel free to re-use code from the previous laboratory sessions, so long as you understand it. You may also discuss the project with other students in the class, but your programs should otherwise be your own original work – this is **not a group project**!

You are **required to submit your code** via an Assignment item on the course's Moodle page, **by the end of the same day as your scheduled laboratory session**, following the instructions provided there. Your code may be cross-checked for plagiarism, so make sure that you do not copy any other student's actual implementation, or base your solution on one that you obtain from another student.

4.2 Managing the limited resource of demonstrator time

During your labs, demonstrators will have a major responsibility of marking your project. This is time consuming, and so you cannot expect to be marked only in the last hour of a lab session. To maximize your opportunity to be marked you should come prepared to the lab session in Week 8 with many elements of your project completed or at least partially working, so that you can ask a demonstrator to mark or look over your solution as early as possible within the lab session.