Elec4622/Elec9722 Laboratory Project 1, 2018 S2

David Taubman

August 22, 2018

1 Introduction

This is the first of three mini-projects to be demonstrated and assessed within the regular scheduled laboratory sessions. This project is due in the Week 9 laboratory session. The project is worth a nominal 10 marks, out of the 30 marks available for the laboratory component of Elec4622/Elec9722. However, there are optional elements which can attract **a lot of potential bonus marks**.

Get ready to put a lot of work into this project, but also to learn a huge amount from it! Success will build confidence, so invest the effort and **use your remaining laboratory sessions wisely**.

In this project, you will implement and explore the implications of various strategies for image resampling. To simplify matters, we consider only two cases here:

1. Reducing the size of an image by a factor of 5/3 in each direction – i.e., zoom-out.
2. Increasing the size of an image by a factor of 5/3 in each direction – i.e., zoom-in.

2 A Brief Review of the Theory

Let $x[\mathbf{n}]$ denote a discrete digital image, which we assume to arise from the sampling of an underlying bandlimited signal $f(\mathbf{s})$. This continuous signal can then be recovered via

$$f(\mathbf{s}) = \sum_{\mathbf{n}} x[\mathbf{n}]\, q(\mathbf{s} - \mathbf{n})$$

where the interpolation kernel ideally satisfies $q(\mathbf{s}) = \operatorname{sinc}(s_1)\operatorname{sinc}(s_2)$.

2.1 Expansion

To expand the image by a factor of 5/3, we simply need to resample $f(\mathbf{s})$ at the locations $\mathbf{s} = \frac{3\mathbf{m}}{5}$, obtaining

$$y_{\times\frac{5}{3}}[\mathbf{m}] = f\left(\frac{3\mathbf{m}}{5}\right) = \sum_{\mathbf{n}} x[\mathbf{n}]\, q\left(\frac{3\mathbf{m}}{5} - \mathbf{n}\right) \tag{1}$$

Resizing is a separable operation. This can be deduced by recognizing that the $q(\mathbf{s})$ interpolation kernel used in the above formula is separable. However, it is also intuitively obvious that we can first expand each image row and then expand each column of the result, or vice-versa. In view of this, we can simplify our analysis (and implementation) by considering the expansion in one dimension only. In this case, equation (1) becomes

$$y_{\times\frac{5}{3}}[m] = \sum_n x[n]\, q\left(\frac{3m}{5} - n\right)$$

In the expanded result, those locations whose indices are multiples of 5 are found from

$$y_{\times\frac{5}{3}}[5m] = \sum_n x[n]\, q(3m - n) = x[3m],$$

since $q(s) = \operatorname{sinc}(s)$ is equal to $\delta(n)$ at integer locations n. This just means that every second input sample aligns perfectly with every fifth output sample – no interpolation is required. The other locations are found from

$$\begin{aligned}
y_{\times\frac{5}{3}}[5m+1] &= \sum_n x[n]\, q\left(3m - n + \frac{3}{5}\right) = \sum_n x[n]\, q\left(3m + 1 - n - \frac{2}{5}\right) \\
&= \sum_n x[n]\, q_{\frac{2}{5}}[3m + 1 - n] \\
y_{\times\frac{5}{3}}[5m+2] &= \sum_n x[n]\, q\left(3m - n + \frac{6}{5}\right) = \sum_n x[n]\, q\left(3m + 1 - n + \frac{1}{5}\right) \\
&= \sum_n x[n]\, q_{-\frac{1}{5}}[3m + 1 - n] \\
y_{\times\frac{5}{3}}[5m+3] &= \sum_n x[n]\, q\left(3m - n + \frac{9}{5}\right) = \sum_n x[n]\, q\left(3m + 2 - n - \frac{1}{5}\right) \\
&= \sum_n x[n]\, q_{\frac{1}{5}}[3m + 2 - n] \\
y_{\times\frac{5}{3}}[5m+4] &= \sum_n x[n]\, q\left(3m - n + \frac{12}{5}\right) = \sum_n x[n]\, q\left(3m + 2 - n + \frac{2}{5}\right) \\
&= \sum_n x[n]\, q_{-\frac{2}{5}}[3m + 2 - n]
\end{aligned}$$

where

$$\begin{aligned}
q_{\pm\frac{1}{5}}[n] &= q\left(n \mp \frac{1}{5}\right) \\
q_{\pm\frac{2}{5}}[n] &= q\left(n \mp \frac{2}{5}\right)
\end{aligned}$$

That is, $q_\sigma[n]$ is the digital filter whose PSF is obtained by delaying (translating to the right) the basic $\operatorname{sinc}(s)$ function by σ and then sampling it at the integer locations. To obtain $y_{\times\frac{3}{5}}[5m+1]$, we convolve x by $q_{\frac{2}{5}}$ and keep only the locations $3m+1$ from the result. Similarly, we obtain $y_{\times\frac{3}{5}}[5m+2]$ by convolving x with $q_{-\frac{1}{5}}$ and keeping only the locations $3m+1$, and so forth.

It is better computationally, and much simpler conceptually, to rewrite the above expressions as inner products. In this case we have

$$\begin{aligned} y_{\times\frac{5}{3}}[5m+1] &= \sum_n x[n]\,\tilde{q}\left(n - \frac{15m+3}{5}\right) \\ &= \sum_n x[n]\,q\left(n - 3m - \frac{3}{5}\right) \quad \text{– noting that } q(s) = q(-s) \\ &= \sum_n x[n]\,q_{3m+0.6}(n) = \langle \mathbf{x}, \mathbf{q}_{3m+1-0.4}\rangle \end{aligned}$$

and, similarly,

$$\begin{aligned} y_{\times\frac{5}{3}}[5m+2] &= \langle \mathbf{x}, \mathbf{q}_{3m+1.2}\rangle = \langle \mathbf{x}, \mathbf{q}_{3m+1+0.2}\rangle \\ y_{\times\frac{5}{3}}[5m+3] &= \langle \mathbf{x}, \mathbf{q}_{3m+1.8}\rangle = \langle \mathbf{x}, \mathbf{q}_{3m+2-0.2}\rangle \\ y_{\times\frac{5}{3}}[5m+4] &= \langle \mathbf{x}, \mathbf{q}_{3m+2.4}\rangle = \langle \mathbf{x}, \mathbf{q}_{3m+2+0.4}\rangle \end{aligned}$$

What this means is that you obtain $y_{\times\frac{5}{3}}[5m+1]$ for any given m by translating the function $q_{-0.4}[n]$ across by $3m+1$ samples and taking the dot product with the input sequence – note that $3m+1$ is the nearest integer location to the desired location of $\frac{3}{5}(5m+1)$. Similarly, you obtain $y_{\times\frac{5}{3}}[5m+2]$ by translating the function $q_{0.2}[n]$ by $3m+1$ and taking its dot product with the input sequence – again, $3m+1$ is just the nearest integer to $\frac{3}{5}(5m+2)$.

In the end, you need to evaluate and store 4 different inner product "kernels," corresponding to $q_{\pm 0.2}[n]$ and $q_{\pm 0.4}[n]$. Then, in every group of 5 output samples, one of them is a direct copy of the corresponding input sample, and each of the other four are obtained by taking inner products with the relevant q kernel, **translated to the sample location which is closest to the one you want**.

Of course, $q_{\pm 0.2}[n]$ and $q_{\pm 0.4}[n]$ are infinite extent sequences, so you will have to window them or apply some other approximation technique in practice. You are now in a position to develop an implementation that is closely related to what you have seen in Lab 2.

Before proceeding, it is very strongly recommended (pretty much mandatory) that you draw up a figure to convince yourself that you thoroughly understand exactly how each group of 5 output samples on a line is derived from input samples via inner products. Your figure should depict exactly what inner products are involved. The figure is useless unless you can actually put windowed sinc sample values into your figure and work out exactly what you would get for the interpolated value at a given location, based on a given set of inputs. If your program has bugs, you will need this figure, since it tells you exactly what to expect at a given location, based on the inputs that may be available. Debugging is all about comparing your expectations against what is actually happening at a given location, as reported live via the debugger. Actually using a debugger is the most efficient way to discover and fix problems, so if all you have ever done is use print statements, now is the time to make sure you know how to use the integrated debugger in the Visual Studio platform (or another platform if you are using something else).

2.2 Reduction

To reduce the image by a factor of $\frac{5}{3}$, we need to resample $f(\mathbf{s})$ at multiples of $\frac{5}{3}$. However, to avoid aliasing, we must be sure first to reduce the bandwidth of $f(\mathbf{s})$ so that $\hat{f}(\boldsymbol{\omega}) \approx 0$ for $\boldsymbol{\omega} \notin [-3\pi/5, 3\pi/5]^2$. Equivalently, we first shrink $f(\mathbf{s})$ by a factor of $\frac{5}{3}$, bandlimit it to $[-\pi,\pi]^2$, and then sample at the integer locations. In this brief review, we shall adopt the former perspective, forming

$$y_{\div\frac{5}{3}}[\mathbf{m}] = g(\mathbf{s})|_{\mathbf{s}=5\mathbf{m}/3}$$

where

$$g(\mathbf{s}) = (f * h_{\text{lpf}})(\mathbf{s})$$

Since $f(\mathbf{s})$ is just a linear combination of shifted copies of the $q(\mathbf{s})$ interpolation kernel, we can obtain $g = f * h_{\text{lpf}}$ by applying the low-pass anti-aliasing filter h_{lpf} directly to the interpolation kernel. That is,

$$g(\mathbf{s}) = \sum_{\mathbf{n}} x[\mathbf{n}]\, q_{\text{lpf}}(\mathbf{s}-\mathbf{n})$$

where

$$q_{\text{lpf}}(\mathbf{s}) = (q * h_{\text{lpf}})(\mathbf{s})$$

In our case, $q(\mathbf{s})$ is ideally $\text{sinc}(s_1)\,\text{sinc}(s_2)$, while $h_{\text{lpf}}(\mathbf{s})$ is ideally $\frac{9}{25}\,\text{sinc}\left(\frac{3s_1}{5}\right)\text{sinc}\left(\frac{3s_2}{5}\right)$, so $q_{\text{lpf}}(\mathbf{s})$ is equal to $h_{\text{lpf}}(\mathbf{s})$.[1]

It follows that

$$y_{\div\frac{5}{3}}[\mathbf{m}] = \sum_{\mathbf{n}} x[\mathbf{n}]\, q_{\text{lpf}}(5\mathbf{m}/3-\mathbf{n})$$

Again, the simplest and most efficient way to implement this is to perform the reduction separably (rows then columns, or vice-versa) and adopt the inner products (output-driven) perspective, as follows. For the even locations of the output, we get

$$\begin{aligned} y_{\div\frac{5}{3}}[3m] &= \sum_n x[n]\, q_{\text{lpf}}(5m-n) = \sum_n x[n]\, \tilde{q}_{\text{lpf}}(n-5m) \\ &= \sum_n x[n]\, q_{\text{lpf}}(n-5m) = \langle \mathbf{x}, \mathbf{q}_{\text{lpf},5m}\rangle \end{aligned}$$

That is, we take the inner product between the input sequence and a copy of q_{lpf} that is translated by $5m$ and sampled at the integer locations. Similarly, for the other locations of the output, we get

$$\begin{aligned} y_{\div\frac{5}{3}}[3m+1] &= \sum_n x[n]\, q_{\text{lpf}}\left(5m+\frac{5}{3}-n\right) \\ &= \sum_n x[n]\, q_{\text{lpf}}\left(n-5m-\frac{5}{3}\right) = \left\langle \mathbf{x}, \mathbf{q}_{\text{lpf},5m+2-\frac{1}{3}}\right\rangle \\ y_{\div\frac{5}{3}}[3m+2] &= \sum_n x[n]\, q_{\text{lpf}}\left(5m+\frac{10}{3}-n\right) \\ &= \sum_n x[n]\, q_{\text{lpf}}\left(n-5m-\frac{10}{3}\right) = \left\langle \mathbf{x}, \mathbf{q}_{\text{lpf},5m+3+\frac{1}{3}}\right\rangle \end{aligned}$$

That is, we need three sets of discrete kernels, $q_{\text{lpf}}(n)$, $q_{\text{lpf},\frac{1}{3}}(n)$ and $q_{\text{lpf},-\frac{1}{3}}(n)$ that are obtained by sampling the stretched sinc function q_{lpf}, shifted by 0 and $\pm\frac{1}{3}$, then each output sample is obtained by

[1] If this is not obvious to you, consider what happens in the Fourier domain. $\hat{q}(\boldsymbol{\omega})$ is equal to 1 inside $[-\pi,\pi]^2$ and 0 elsewhere. $\hat{h}_{\text{lpf}}(\boldsymbol{\omega})$ is equal to 1 inside $[-3\pi/5, 3\pi/5]^2$ and 0 elsewhere. Consequently, $\hat{q}(\boldsymbol{\omega})\,\hat{h}_{\text{lpf}}(\boldsymbol{\omega}) = \hat{h}_{\text{lpf}}(\boldsymbol{\omega})$.

translating the relevant kernel to the nearest integer location to the desired one – $\frac{5}{3}(3m+1)$ is closest to integer location $5m+2$, and $\frac{5}{3}(3m+2)$ is closest to integer location $5m+3$ – and taking the inner product between the input sequence and the shifted kernel.

It is worth sketching what is happening in the above equations on paper. In particular, you will find that everything is remarkably simple. All we are doing in every case is translating an appropriate sinc function so that it is centred at the location where we want to obtain an output sample, whereupon we take the inner product between the input samples and the samples of the translated sinc function.

Again, since sinc functions have infinite extent, you will need to window them or use some other approximation technique. Be sure to remember to use a window that has the same centre of symmetry as the underlying continuous sinc function that will be sampled.

3 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 will not be marked for your work unless it is organized in this way.

You should also rely much more heavily on viewing images using `mi_viewer` than the Windows previewer. There are lots of reasons for this, but the obvious ones are that you can launch multiple copies to view multiple images at once, and you can zoom in and inspect individual pixel values with ease. Ultimately, it is a great deal faster and less confusing to execute your programs and view your images directly from the DOS prompt, than by using mouse clicks – assuming you can type. Remember that the up and down arrows allow you to scroll through and edit previous commands easily. Remember also that you can use the tab key to expand file names, program names, etc., so you rarely have to type everything in full.

One thing you need to be prepared for is that it may be difficult at first to know whether your program is working correctly. This is a real world dilemma that you need to learn how to address. Just because your program compiles and produces something which looks like an image does not mean it is doing the right thing. So you need to have a good feeling for what the result should look like, based on your understanding of the fundamental concepts.

- To debug your program, you may find it useful to process very tiny images with only a few samples, so you can check that the output is as expected; you can use something like `"mi_pipe2 -i image.bmp -o tiny.bmp -form bmp ::  crop_n_shuffle -crop 0 0 16 16"` to create a 16x16 image by cropping a much bigger one. You can also read the sample values directly within the "mi_viewer" application by zooming in (z accelerator) and holding the mouse button down.

- It is a good idea to rely upon the theory you have learned to verify your programs. For example, you know that a good resolution reduction algorithm should be shift invariant, or nearly so. If this is the case, you should not observe any kind of modulation or pattern in the output which did not exist in the input – pay special attention around oriented edges, which can be understood as a single feature that consistently shifts from row to row or column to column by a non-integer amount.

- You can also use apply your interpolation program to the output of your reduction program and vice-versa. In each case you will obtain an image of the same size as the original (or a little larger, in which case you can crop it down), so you can use the Media Interface tools to compare the resulting image with the original – this is a powerful way to check for any shift variance issues. One useful tool is the Media Interface "image_arith" module, which can be used to form the difference between two images as follows:

  - mi_pipe2 -in1 image1.bmp -in2 image2.bmp :: image_arith -add 1 -1

It should be apparent that 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.

4 Tasks

Task 1: (5 marks **+ up to 3 possible bonus marks**) Write a program which accepts a BMP image as input and writes out a reduced image, having dimensions $\lceil \frac{3}{5}W \rceil \times \lceil \frac{3}{5}H \rceil$. Notice that you should round the dimensions up to the nearest integer (ceiling function). Your program should be based upon the method of Section 2.2, with windowed sinc filters. You should adopt a Hanning window. Here are some additional specifics:

1. Your program should accept a **command-line argument** H (loosely interpreted as the window half-length) such that the filter PSFs (or inner product kernels) q_{lpf} and $q_{\text{lp},\pm\frac{1}{3}}$ all have their non-zero values falling within a $(2H+1)\times(2H+1)$ region of support. Your program should function correctly for values of H in the range 0 through to 14 at least, so that you can demonstrate the effect of different filter complexities. The special value $H = 0$ means, of course, that your inner products all involve exactly one coefficient – if your program is any good, this will be equivalent to nearest neighbour interpolation – check that it works correctly by looking at the numerical values with **"mi_viewer."**
2. Windowing may impact the DC gain of the filter somewhat. Be sure to renormalize your filters so that the DC gain is always 1 in this application.
3. It is best to apply a continuous window function directly to the sinc function $q_{\text{lpf}}(s)$, then shift and sample the windowed result, so that you do not accidentally introduce further shifts due to a window that has a different centre of mass to the sinc function. Draw sketches and figure out what continuous window parameter τ should be selected for a given value of the integer parameter H that your program accepts on the command-line.
4. You are free to use either a floating-point or a fixed-point (integer) implementation, but you will probably find a floating-point implementation to be easier for this exercise.
5. Remember to apply a suitable boundary extension method prior to filtering.
6. **Up to 3 bonus marks** may be awarded for efficient implementation of this task. Ideally, your implementation should strive to perform as few computations as possible, while still correctly implementing the reduction filter for the specified window size. You should consider separability as an important basis for minimizing the complexity. You may also consider the option of fixed

point processing with 16-bit sample and coefficient precisions. To gain these bonus marks you will need to demonstrate actual measured improvements in processing speed.

Task 2: (2 marks) Write a program which accepts a BMP image as input and writes out an expanded image, having $\frac{5}{3}$ times the number of rows and columns. The new dimensions should be $\lceil\frac{5}{3}W\rceil \times \lceil\frac{5}{3}H\rceil$. Your program in this part should be based on bi-linear interpolation (see Chapter 3 of your lecture notes). Note that bi-linear interpolation is similar to sinc-interplation, in that you are finding the nearest location in the input image to the desired one, and taking the inner product of a small neighbourhood of samples around that location with the coefficients of a kernel that depends on the shift amount. In this case, however, the kernel only has 4 coefficients and the formula is very simple. You might like to go back over the Week 5 lecture to understand what it means to interpolate an image using the bi-linear kernel, rather than a sinc kernel. You should look for artefacts in the interpolated result that might reveal the effects of aliasing, even though it will not be a very strong effect for most inputs.

Task 3: (3 marks + **up to 2 possible bonus marks**) Write a program which accepts a BMP image as input and writes out an expanded image, having $\frac{5}{3}$ times the number of rows and columns, as above. Your program in this part should be based on the method of Section 2.1, with windowed sincs. Here are some additional specifics:

1. The half-window size parameter H should be a **command-line argument** to your program, so that you can demonstrate the effect of different filter lengths. The meaning of H is, as before, that all inner products (if implemented non-separably) should involve kernels with a region of support that is limited to $(2H+1)\times(2H+1)$. Your program must work for values of H in the range 0 through 7, where $H=0$ again corresponds to nearest neighbour interpolation – if you have implemented things correctly, this does not need to be treated as a special case.
2. Windowing may impact the DC gain of the interpolation kernels somewhat. Be sure to renormalize your coefficients so that the DC gain is always 1.
3. Remember to apply a suitable boundary extension method prior to filtering.
4. **Up to 2 bonus marks** may be awarded for efficient implementation of this task. Again, you should consider separability and you will need to demonstrate actual measured improvements in speed. You may also be rewarded for efficient memory utilization in the implementation of your expansion operator.

Task 4: (**Up to three bonus marks**) Write a program which accepts two BMP images $x[\mathbf{n}]$ and $y[\mathbf{n}]$ as input, generates a difference image $128+\frac{1}{2}(x[\mathbf{n}]-y[\mathbf{n}])$ as output, and also computes the mean error and mean squared error (MSE) between the input images. If the images are each of size $M\times N$, these quantities are

$$\text{Mean Error } = \frac{1}{MN}\sum_{n_1=0}^{M}\sum_{n_2=0}^{N}(x[\mathbf{n}]-y[\mathbf{n}])$$

and

$$\text{MSE } = \frac{1}{MN}\sum_{n_1=0}^{M}\sum_{n_2=0}^{N}(x[\mathbf{n}]-y[\mathbf{n}])^2$$

From the MSE, you should derive the peak signal to noise ratio (PSNR), which is defined by

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\text{MSE}}$$

for 8-bit sample values.

Note: While this task is simple, you will need to deal with images that do not have identical dimensions to perform all the investigations below. Where the input images have different dimensions, your program should ignore extra rows on the bottom or columns on the right of the larger image.

1. **1 bonus mark:** Arrange for your program to print the Mean Error, MSE and PSNR for each colour plane, as well as generating the difference image.
2. **1 bonus mark:** Use your program to investigate the difference between an original image and one obtained by first expanding it and then reducing it back to its original size. Compare the performance of the bi-linear and windowed sinc expansion algorithms represented by Tasks 2 and 3. Also, determine the impact of the window sizes. **To receive these bonus marks you must be prepared to explain what you observe.**
3. **1 bonus mark:** Use your program to investigate the difference between an original image and one obtained by first reducing it and then expanding it back to its original size. Again, study the effect of window size and bi-linear vs. windowed sinc expansion. **To receive these bonus marks you must be prepared to explain what you observe.**

5 Assessment

You should not rely upon implementing this project within the scheduled laboratory sessions. You will have an opportunity to do this in the Week 7 and even Week 9 laboratory sessions, but it is recommended that you aim to have some of the project implemented prior to Week 7 so that you can get feedback from the demonstrators prior to final assessment. In any event, you must be prepared to demonstrate your work by Week 9 and be assessed individually at any time during the last two hours of that session – **you cannot expect demonstrators to mark your work only during the final hour of the laboratory session**.

In order to obtain the full marks for any given task, you must have a working program to demonstrate and you must be able to explain how it works and **answer questions very quickly**!! We will not be able to wait around while you fiddle with running your program from the debugger or viewing images with the Windows previewer, so have things set up for easy validation by the course staff and **use "mi_viewer"** to show your images directly from the command-line, allowing multiple images to be viewed simultaneously and individual pixel values to be read-off by the marker with simple mouse clicks.

You may feel free to re-use code from Labs 1 to 3, so long as you understand it. You may also discuss the project with other students in the class, but **your programs should be your own original work**. As with assessment tasks in other subjects, UNSW regards **plagiarism as a serious offense**; for a first offense, the penalty would be loss of all marks for the project.