Fast Image Enhancement by Controlled Blur and Substitution

Kevin Pulo, `mailto:kev@zip.com.au`

June 1999

Abstract

Image enhancement is usually a computationally time consuming process. Our aim was to implement an algorithm to perform a fast and quantitative sharpening of images, with such applications as real-time image enhancement.

1 Introduction

Every optical system has a limited, finite resolution and introduces noise into the signals it measures. However, these problems can be addressed in a quantitative fashion to enhance the recorded image so that it is closer to the actual image, in both a quantitative and qualitative sense. These optical systems are usually high resolution optical microscopes, often confocal microscopes.

1.1 Theory

An optical system's finite resolution is described by its *optical transfer function* or *OTF*

$$o(-\Delta\vec{x}) \tag{1}$$

This represents the signal contribution of a point located an offset $\Delta\vec{x}$ from the point the system is measuring. For a perfect system, $o(-\Delta\vec{x}) = \delta(\vec{x})$, where $\delta(\vec{x})$ is the Dirac-Delta function. However, for real systems of finite resolution, $o(-\Delta\vec{x})$ can be measured experimentally by recording the image $i(\vec{x})$ of a very small, point-like source, acting essentially as a baseline signal. The OTF is then given by $o(-\Delta\vec{x}) = i(\vec{x}_0 + \Delta\vec{x})$ where $\vec{x}_0$ is the (known) position of the point-like source.

In addition, as the OTF depends only on the physical nature of the instruments, it can be analysed to give a theoretical OTF for the system. This is as good as the OTF obtained experimentally and as shown in Section 2.1.2 has several practical advantages over an experimental OTF.

Equation 1 has been defined in terms of $-\Delta\vec{x}$ because this means that any measured image $i(\vec{x})$ derives from the original ("true") image $i_0(\vec{x})$ by convolution with the optical transfer function.

$$i(\vec{x}) = i_0(\vec{x}) \otimes o(-\Delta\vec{x}) \tag{2}$$

Now when working in Fourier space, we can use the identity that convolution is equivalent to multiplication of Fourier transforms to get

$$\mathbf{F}(i) = \mathbf{F}(i_0) \cdot \mathbf{F}(o) \tag{3}$$

where the boldface $\mathbf{F}$ indicates a Fourier transform and the Fourier transform of the optical transfer function $\mathbf{F}(o)$ is sometimes referred to as the *point spread function* or *psf*. Quantitatively sharpening the image $i(\vec{x})$ to obtain $i_0(\vec{x})$ involves the deconvolution of equation 2, which now becomes the inversion of equation 3

$$\mathbf{F}(i_0) = \frac{\mathbf{F}(i)}{\mathbf{F}(o)} \tag{4}$$

Unfortunately, real optical systems also suffer from the addition of noise by the instruments involved in the system. We modify equation 2 to account for the noise n_p due to a photon p and the optical instrument noise n_o and obtain

$$\begin{aligned} i &= (i_0 + n_p) \otimes o + n_o \qquad (5) \\ &= i_0 \otimes o + (n_p \otimes o + n_o) \\ &= i_0 \otimes o + n \end{aligned}$$

for the collected noise term $n = n_p \otimes +n_o$. However, n cannot be known and so calculating Equation 5 is no longer possible. Further, considering small $\mathbf{F}(i)$ in Equation 4 we see that neglecting n will introduce major disturbances and artifacts in the restored image.

1.1.1 Exact image enhancement

The sophisticated approach to handle this noise problem is to deconvolute the image under constraints, usually penalty functions for high frequencies in the restored image. Unfortunately, this is computationally expensive — it requires heuristics and there is no scalable or otherwise efficient algorithm to find acceptable heuristics or perform the deconvolution itself.

1.1.2 Our approach

Our approach trades accuracy for speed, suitable for applications where some accuracy can be sacrificed to gain speedups in the image enhancement.

The most notable such application is that of real-time image enhancement, where the user of a microscope (or other optical imaging instrument) desires real-time feedback on the microscope's subject as it is being scanned. Here, an algorithm's main priority is to be fast, to give the user feedback as soon as possible, without spending a lot of time restoring the image. Once the user is satisfied with this preliminary scan (and the fast image enhancement associated) they may then scan the subject at the highest resolution and possibly with an exact image enhancement algorithm, knowing that while it may take some time, they have an idea of what the result should be like (in terms of quality).[1]

2 Method

2.1 Algorithm description

The algorithm consists of blurring the measured image $i(\vec{x})$ with a regularizing smoothing function $r(\vec{x})$. There are several possible choices of the function $r(\vec{x})$, however this project isn't concerned with finding an optimal $r(\vec{x})$; we will use the simple unitary Gaussian function in three dimensions[2]

$$\begin{aligned} r(\vec{x}) &= e^{-|\vec{x}|^2} \qquad (6) \\ &= e^{-x^2-y^2-z^2} \end{aligned}$$

We now obtain the smoothed, measured image

$$\begin{aligned} R &= i \otimes r \\ &= ((i_0 + n_p) \otimes o + n_o) \otimes r \qquad (7) \end{aligned}$$

Here we introduce the imperfection that lowers accuracy but increases speed, and it is to assume that all the noise was brought in before the convolution with o (specifically the n_o instrument noise term). Based on this assumption, Equation 5 is modified to become

$$i = (i_0 + n_p + n_o) \otimes o \qquad (8)$$

Equation 7 now becomes

$$\begin{aligned} R &= (i_0 + n_p + n_o) \otimes o \otimes r \qquad (9) \\ &= i \otimes r \end{aligned}$$

Observing that this is simply the measured image after being smoothed. The actual method used to calculate the smoothed image R will be examined in Section 2.1.1. We now obtain an approximation to the true image by deconvoluting Equation 9 by the smoothed optical transfer function $o \otimes r$

$$\mathbf{F}(i_0) \cong \frac{\mathbf{F}(R)}{\mathbf{F}(o \otimes r)} \qquad (10)$$

[1] Further, hardware implementations of algorithms often yield speed increases of approximately an order of magnitude. This algorithm may not be suited to having a full hardware implementation (as it requires substantial amounts of memory for reasonable resolution images), but parts of it may be suited to hardware implementations.

[2] Which also has the additional feature of being its own Fourier transform, that is, $\mathbf{F}(f) = f$ for $f(\vec{x}) = e^{-|\vec{x}|^2}$. More generally, the Fourier transform of a Gaussian function is also a Gaussian function (with different constants).

Now we have the problem that Equation 10 becomes unreliable when $|\mathbf{F}(o \otimes r)|$ becomes small. The sophisticated way to avoid this problem is to use an averaging filter. A suitable averaging term is $\alpha = \frac{1}{1+w_0 \cdot |\mathbf{F}(o \otimes r)|}$ for w_0 an arbitrarily suitable weighting term. $\sum \mathbf{F}_\sigma(i_0)$ is the sum of the neighbourhood σ in i_0, where usually σ is taken as the 4-element neighbourhood of $\mathbf{F}(R)$ and $\mathbf{F}(o \otimes r)$. The averaging procedure is then

$$\begin{aligned}
\mathbf{F}'(i_0) &= (1-\alpha)\frac{\mathbf{F}(R)}{\mathbf{F}(o \otimes r)} + \frac{\alpha}{4}\sum \mathbf{F}_\sigma(i_0) \text{ for } \mathbf{F}(o \otimes r) \neq 0 \qquad (11) \\
\mathbf{F}'(i_0) &= w_0 \cdot \mathbf{F}(R) + \frac{1}{4}\sum \mathbf{F}_\sigma(i_0) \text{ otherwise}
\end{aligned}$$

However, Equation 11 is only defined when $\sum \mathbf{F}_\sigma(i_0)$ exists, specifically, $\sum \mathbf{F}_\sigma(i_0) \neq 0$. So we sum the averaged elements instead, giving the recurrence relation

$$\begin{aligned}
\mathbf{F}'(i_0) &= (1-\alpha)\frac{\mathbf{F}(R)}{\mathbf{F}(o \otimes r)} + \frac{\alpha}{4}\sum \mathbf{F}'_\sigma(i_0) \text{ for } \mathbf{F}(o \otimes r) \neq 0 \qquad (12) \\
\mathbf{F}'(i_0) &= w_0 \cdot \mathbf{F}(R) + \frac{1}{4}\sum \mathbf{F}'_\sigma(i_0) \text{ otherwise}
\end{aligned}$$

This definition requires a recursive method to be used, such as flood-filling.

In order to simplify things somewhat, we used the observation that the most common pattern under which $\sum \mathbf{F}_\sigma(i_0) \neq 0$ is for thin, two dimensional spherical shells[3]. Thus, the flood-filling technique is excessive and would rarely come into play in normal images. Our solution will sacrifice some image quality in order to decrease the complexity of this stage of the algorithm.

We adopt basically the same strategy as above, except that we define Equation 11 to be valid for $\sum \mathbf{F}_\sigma(i_0) \geq \epsilon$, for some ϵ suitably close to 0. We then process the image iteratively, not recursively, and if a position with $\sum \mathbf{F}_\sigma(i_0) < \epsilon$ is encountered we simply continue by assigning $\sum \mathbf{F}_\sigma(i_0) = \epsilon$ and continuing as in Equation 11.

2.1.1 On smoothing

In the calculations detailed above (Equations 6 to 10), R was obtained by convoluting the measured image by the smoothing function r, with the implication that it would be calculated by a Fourier transform and subsequent point-wise multiplication of the two terms. However, if this were the case we would have the subtle problem of Equation 10 being

$$\begin{aligned}
\mathbf{F}(i_0) &\cong \frac{\mathbf{F}(R)}{\mathbf{F}(o \otimes r)} \\
&= \frac{\mathbf{F}(i) \cdot \mathbf{F}(r)}{\mathbf{F}(o) \cdot \mathbf{F}(r)}
\end{aligned}$$

[3]For the three dimensional images that we are using. Naturally if we were dealing with two dimensional images we would expect to find thin rings.

$$= \frac{\mathbf{F}(i)}{\mathbf{F}(o)}$$

That is, it has reduced back to Equation 4, where the smoothing function r is not needed.

However, the reason for r's existence is that we can smooth i in real space, not Fourier space. This is achieved using a mask operation on i, for example, using the convolution matrix of

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 5 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

and subsequently scaling the pixel values by $\frac{1}{13}$, or, for a more aggressive smoothing, the convolution matrix of

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 5 & 5 & 5 & 1 \\ 1 & 5 & 44 & 5 & 1 \\ 1 & 5 & 5 & 5 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

scaling this time by $\frac{1}{100}$.

Again, this procedure will decrease the accuracy somewhat, but it will increase the speed of the algorithm substantially as this method is localized for each point, requiring only a constant amount of computational time per point, which is faster than any Fourier transform (refer to Section 4.1 for a complete analysis of the time complexity of the algorithm).

2.1.2 On the optical transfer function

The term $\mathbf{F}(o \otimes r)$ as it appears in Equation 10 (and various others) can be further optimised if a theoretical OTF is used. An experimentally obtained OTF o requires its Fourier transform to be calculated in order for it to be convoluted with the smoothing function r[4], this requires both memory and time. However, if a theoretical equation is obtained for o, then the entire term $\mathbf{F}(o \otimes r)$ may be evaluated theoretically. This will give an equation that can be used in the algorithm, in order to deconvolute Equation 10. Thus, a substantial amount of time and memory can be saved by using a theoretically computed OTF. Refer to Section 2.2 for details of what OTF was used.

2.1.3 On weight functions

There are several choices of weight functions, w_0. The most basic, that we have opted to use, is that of the neighbourhood of 4 most adjacent positions, as shown by the dark grey in Figure 1, with each position being weighted $\frac{1}{4}$ relative to the other positions.

[4] Or at the very least, a mask-based smoothing operation as detailed in Section 2.1.1.

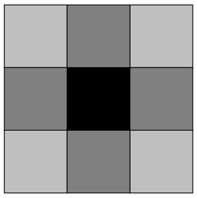

Figure 1: The neighbourhood of 4 most adjacent positions (dark grey) and 4 diagonal (light grey) positions to the central (black) position.

Another possibility is that of taking the most adjacent positions and the diagonally adjacent positions, as shown by the dark and light grey in Figure 1. Here, each of the most adjacent positions would have a relative weight of $\frac{1}{6}$ and each of the diagonally opposite positions would have a relative weight of $\frac{1}{12}$, this way the sum of the weights is still unity and no scaling of the central pixel value occurs.

Note that the 26 element, three dimensional neighbourhood is not necessarily a better neighbourhood to take than the above 8 element two dimensional neighbourhood. While all 26 elements are still local to the central position, the microscopes that create these images usually do so in a layered fashion, that is, they scan a layer of fixed z at a time. Thus weighting functions are better taken where all the elements considered are within a single, two dimensional layer of fixed z.

2.1.4 Algorithm summary

The overall algorithm is now summarised into its most salient steps.

- The measured image is smoothed in real space using a mask operation (Section 2.1.1).
- The Fourier transform of the smoothed image is found.
- The Fourier transform of the deconvolution term, $\mathbf{F}(o \otimes r)$ is found (Section 2.1.2).
- The smoothed image is then deconvoluted in Fourier space according to the required weightings (Section 2.1.3 and Equations 10 to 12).
- Finally, the inverse Fourier transform is taken to yield the restored image in real space.

2.2 Implementation

The program was written in ANSI C, using the freely available FFTW package [Frigo and Johnson, 1999]. FFTW is, for all intents and purposes, the fastest freely available package to calculate *Discrete Fourier Transforms* (*DFTs*). Further, it is easy to install and use, giving accurate results quickly.

Unfortunately, the exact optical transfer function for the equipment used to obtain the data files wasn't available, and so a general OTF was used. This was $\frac{\sin r}{r}$ for the radial direction r in the xy plane and $J_0(z)$ in the z direction, where J_n is the Bessel function of the first kind of order n (n a positive integer)[5]. There are many other theoretical OTFs that could have been used, however, they lie outside the scope of this project and the general OTF described above was deemed to be adequate.

The file format used was a type of raw file, with a 76 byte header followed by the image data. The header consisted of (amongst other things) three big-endian words describing the x, y and z dimensions. Following the header was xyz bytes of pixel data, one byte per pixel, giving 256 possible pixel values[6]. The pixels were stored in column-major order, that is, quickest traversal of x, y and z, and were easily converted to the row-major order required by FFTW.

Some constants were required for this project, and they were selected under the advice of those in the field (see Section 5), since finding optimal values was outside the scope of this project. ϵ was chosen to be 2% of the available pixel values, which is a value of 4 (out of 256)[7]. w_0 was chosen to be unity, as this made several aspects of the implementation easier and should not have degraded performance (either image quality or speed) noticeably.

In addition to the main program, a utility was written to take a two dimensional slice of fixed z value and output it in a format suitable for *gnuplot*, as shown in Figure 2, for example.

3 Results

Three data files were used, with sizes of 256k points, 5952k points and 13568k points[8]. Unfortunately, time restraints meant that these files couldn't have an exact image enhancement algorithm (as described in Section 1.1.1) run on them, and so these results cannot be compared to an exact algorithm as we would have liked.

In addition, technical problems prevented imaging of an approximate point source to be used as a basis for the OTF, and so unfortunately results for an experimental OTF couldn't be obtained.

[5] Implemented by taking the first 100 terms of the infinite sum describing J_n, and precomputing the values of J_n for the values of z that would be required. The sum used was

$$J_n(x) = x^n \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m}}{2^{2m+n} m!(n+m)!}$$

as derived in [Kreyszig, 1993, Section 5.5, page 227].

[6] The file format actually has an option to use (big-endian) word length pixels (giving 65536 possible pixel values), however none of our data files used this feature.

[7] This was also observed to be a natural choice, as the images produced by the equipment have low level noise values of approximately this value where no signal is received.

[8] Here, 1k = 1024, as is standard for computing applications.

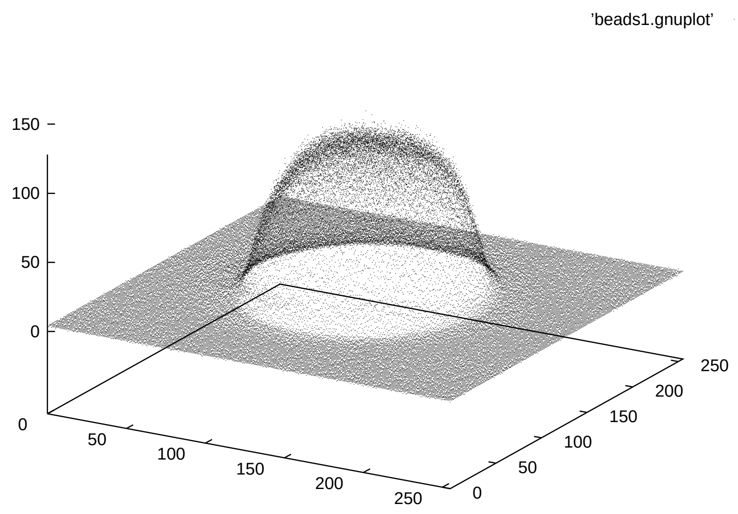

Figure 2: A slice of the `beads1.pic` file, showing a central cross-section of the bead.

3.1 Image quality results

In order to assess the qualitative performance of the algorithm, *gnuplot* was used to plot slices of the three dimensional data. Figure 2 illustrates a cross-section of a bead from the original measured image. With the limited display techniques available to this project, the resultant images weren't able to be properly analysed for their correctness or accuracy. Specifically, when displayed in gnuplot (as in Figure 2) the resolution is too coarse and the angle of view awkward — it is too hard to find the differences between the resultant images and the original images, and only single slices can be analysed at a time. However, the results do appear to be promising, and it is expected that further analysis would confirm that the images have not lost much accuracy.

In addition, if the exact image enhancement algorithms were available, then a direct comparison of the two sets of resultant images could be made, to directly highlight any areas of the images that contained imperfections.

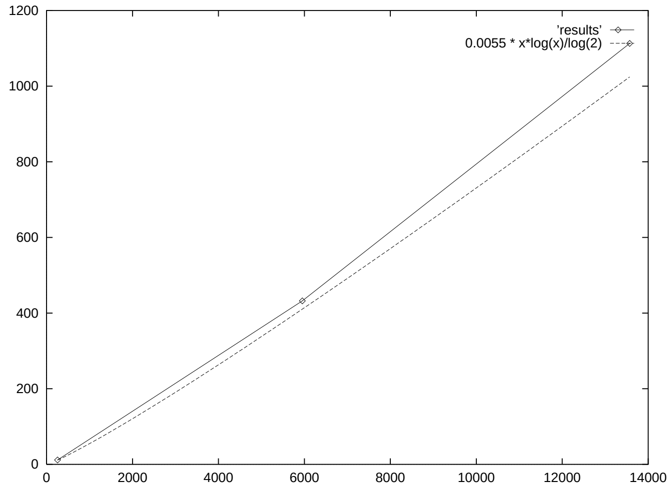

Figure 3: The CPU time taken to run the program. Horizontal axis is dataset size in kilobytes, vertical axis is CPU time in seconds.

3.2 Speed results

Figure 3 details the CPU time[9] required for each of the sample data files, as a function of the number of data points in the data file. These results are consistent with the analysis given in Section 4.1, where the algorithm appears to have a time complexity constant of approximately 0.0055, i.e. time complexity $0.0055n \lg n$.

Despite the lack of speed results for an exact image enhancement algorithm, we are confident that the speeds obtained here are promising and reflect the anticipated speed increases as outlined in Section 2.1. Exact image enhancement algorithms require at best exponential time (that is, for n inputs, time of the order of e^n) and as evidenced here, our algorithm performs substantially better than exponential time.

[9]Note that this is the CPU time required to do the calculations. The actual wall-clock time taken for the program to run was substantially higher than these figures, due to the large amounts of IO required (and often also virtual memory swapping). The machine used to run these tests was an Intel Celeron 300Mhz (128kb CPU cache, overclocked to 450MHz) with 128Mb RAM running Linux.

4 Discussion

4.1 Time complexity and scalability analysis

The DFT (see Section 2.2) of an array of n elements can be computed in $\Theta(n \lg n)$ time[10] using the Fast Fourier Transform (FFT) method as derived in [Cormen et al, 1997, Sections 32.2 and 32.3] and as employed by FFTW. To (point-wise) multiply or divide two arrays each of size n together requires $\Theta(n)$ time. Thus each of the convolutions takes $\Theta(n)$ time in Fourier space, and $\Theta(n \lg n)$ time to transform between Euclidean and Fourier space. To perform the smoothing requires a constant amount of time (e.g. 8 elements or 24 elements for a 3x3 or 5x5 matrix respectively) for each pixel, thus this step also requires $\Theta(n)$ time. This is similar to the weighting step, which considers 8 neighbours, thus also having time complexity $\Theta(n)$. Therefore our entire algorithm has time complexity of $\Theta(n \lg n)$, which scales well[11] as $n \to \infty$.

4.2 Conclusion

While several features of this project were lacking (most notably the exact image enhancement algorithms for comparison, see Section 3), the overall outcome appears to be positive. The algorithm appears to have worked with some degree of success, yielding what is believed to be reasonable quality images in a relatively short amount of time. With further work devoted to verifying the algorithm's validity when compared to exact image enhancement algorithms, and further work optimising the algorithm, it could indeed become very versatile where fast image enhancement is preferred over slow, yet exact image enhancement algorithms.

5 Acknowledgements

I would like to thank Dr Felix Margadant (mailto:felix@margadant.ch) and Ms Carol Cogswell (mailto:c.cogswell@physics.usyd.edu.au) of the Department of Physical Optics in the School of Physics, University of Sydney (http://www.physics.usyd.edu.au) for their help and assistance with this project.

This project utilised the resources of the Sydney Regional Visualisation Laboratory (Sydney Vislab http://www.vislab.usyd.edu.au), which is supported by the Australian Research Council.

[10] Here we are using Θ-notation (related to big-Oh notation), which denotes the set of functions $\{f(n)\}$ that are asymptotically bound by $g(n)$

$$\Theta(g(n)) = \{f(n) : \exists c_1 > 0,\ c_2 > 0,\ n_0 > 0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n)\ \forall n \ge n_0\}$$

Also, in our notation $\lg x = \log_2 x$.

[11] Compared to the exact solutions which have exponential time complexity.

References

[Frigo and Johnson, 1999] M. Frigo and S. G. Johnson. *FFTW User's Manual, version 2.1.1.* MIT Laboratory for Computer Science, Massachusetts Institute of Technology, 1999.

[Cormen et al, 1997] T. H. Cormen, C. E. Leiserson and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill and MIT Press, 1997.

[Kreyszig, 1993] E. Kreyszig. *Advanced Engineering Mathematics, Seventh Edition.* John Wiley and Sons, New York, 1993.

[Bates and McDonnell, 1986] R. H. T. Bates and M. J. McDonnell. *Image Restoration and Reconstruction.* Clarendon Press, Oxford, 1986.