Fast approximate defocus deconvolution

Daniel Williams*

ABSTRACT

I present a fast deconvolution algorithm for the real-time deblurring of images degraded by defocus blur. The proposed algorithm performs significantly faster than other popular methods tested, allowing us to deblur 1080p video at 122 fps (364 fps excluding disk I/O) and 4K video at 32 fps (99 fps excluding disk I/O). It has a simple, self-contained implementation, requiring no Fourier transforms or sophisticated libraries. Implementations in Java, OpenCL, and OpenGL are provided and compared against other popular deconvolution algorithms. When compared to unsharp masking, it shows both higher accuracy and performance at large blur radiuses. Additionally, for one-dimensional noise-free signals, the algorithm is proven to converge exactly to the original un-blurred signal. Code is available at https://github.com/dwilliams1114/fast-deblur.

Keywords: deblurring, deconvolution, signal processing, image restoration, real-time, GPU acceleration

1. INTRODUCTION

Image deconvolution (or deblurring) is often considered a time-consuming computation. In this paper, we focus on maximizing the performance of deconvolution rather than optimizing the accuracy.

The degradation of an image in a simple imaging system may be formulated as the following:

$$b = f * g + s$$

where b is the blurred image, f is the ideal image, g is the point spread function (PSF), s is sampling noise, and $*$ denotes convolution. One common case of this degradation is that of an out-of-focus blur (called *bokeh* in photography). In this case, the point spread function, g, is disk-like (Figure 1). We focus our attention on this case. Our goal is to compute f given only the radius of g. We will assume that the added noise, s, is small.

Many methods have been proposed to solve such a problem. Among the most popular are Richardson-Lucy [1,2] and Wiener deconvolution [3]. Richardson-Lucy is a relatively simple, parallelizable, slow but accurate, iterative algorithm that operates in the spatial domain. In contrast, Wiener deconvolution is a fast but complex fast-Fourier-transform (FFT)-based approach that operates in the frequency domain. Both algorithms are applied in a wide range of applications including astronomy and microscopy. Despite their popularity, neither algorithm performs quick enough for real-time use except for small images.

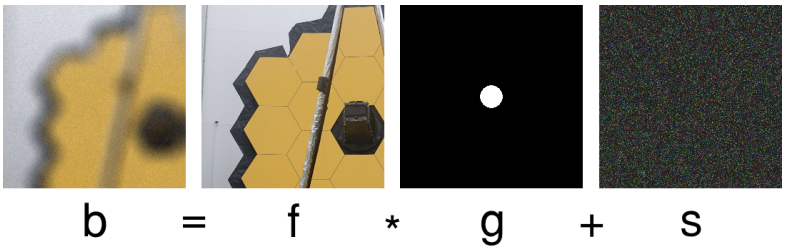

Figure 1. Basic model of defocus-blurred image. Observed degraded image (b) is created by convolving the ideal image (f) with a disk-like point spread function (g) and adding noise (s).

Another (very) popular technique is unsharp masking. Unsharp masking is a standard feature in most image manipulation software and is fast and simple enough that is it commonplace on smartphones, cameras, and other portable and embedded devices. An unsharp mask can be described by the following:

$$f = b + (b - \text{blur}(b))\,\alpha$$

*Independent researcher. Email: danieljwilliams1114@yahoo.com

where b is the original image, f is the final image, and α is a tuning factor for adjusting the intensity of the effect. Unsharp masking is not typically considered to be a deconvolution algorithm since it does not perform the inverse of convolution. However, it is used to improve the "sharpness", "clarity" or "apparent detail" of an image. Unlike many deconvolution algorithms, unsharp masking is easily performed in real-time on modest hardware. It is for this reason that we compare our proposed algorithm against it in the Results.

Our method (which we'll call FAST-METHOD) is intended to give a high-speed approximation to slower deconvolution methods while maintaining useful quality. It is simple to implement, easy to parallelize, fast enough to process high-resolution video faster than real-time, and accurate enough to recover useful structure in an otherwise out-of-focus photograph.

Although we have only mentioned image deblurring, these algorithms (including ours) also operate on one-dimensional signals. We'll focus on the two-dimensional case since it is more computationally expensive, but include analysis of our method in one dimension for completeness.

2. RELATED WORK

A multitude of deconvolution (and deblurring) methods have been proposed. [4], [5], [6], and [7] together give a wide survey of techniques and results. We focus on three key qualities of deblurring methods which are particularly applicable to practical implementation: speed, restoration quality, and sophistication of implementation.

Perhaps the two most well-known methods are Richardson-Lucy and Wiener deconvolution. Richardson-Lucy is straightforward to implement and produces high-quality results given plenty of time. Wiener deconvolution is significantly quicker and still relatively simple, given that an efficient 2D-FFT library is provided, but typically fails to meet the quality of iterative methods.

[8] introduces a particularly fast iterative algorithm which uses the 2D-FFT at each iteration. In [9], this algorithm is optimized for a GPU and achieves 6 fps at 1900x1266 on an NVIDIA GTX 260. [10] proposes a different iterative method whose CPU-only performance essentially matches the CPU-only performance of [8], but without the need for an FFT between iterations.

Non-iterative and non-FFT based approaches have also been proposed. [11] attempts to sharpen fluorescence microscopy imagery by essentially relocating pixels up the image gradient. We mention their method here because of its extreme simplicity.

Some of the fastest deblurring techniques in modern literature are deep-learning-based methods. This is in part due to the prevalence of efficient deep-learning tools. Among the fastest are [12] achieving 8 fps, [13] achieving 19 fps, [14] achieving 25 fps, [15] achieving 58 fps, and [16] achieving 100 fps, all at 1280×720 resolution, (the latter using four NVIDIA 1080Ti's). Modern smartphones, however, may capture images in ultra-high-definition (UHD) and provide significantly less compute power. For UHD images, [17], [18], and [19], require 26.76, 28.41, 31.62 seconds per image respectively according to [4]. This is far from real-time, indicating that fast deblurring of UHD images is still an open problem. The aforementioned approaches are generally focused on motion deblurring tasks (such as those provided by the GoPro dataset [18]), and so cannot be easily compared to our method which only handles defocus deblurring.

Another category of deep-learning based techniques focus specifically on the task of single-image defocus deblurring. [20] achieves 2 fps at 1680×1120 resolution with an NVIDIA Titan X, and [21] achieves 71 fps at 1280×720 resolution with an RTX 3090.

These deep-learning based techniques can hardly be considered simple, requiring machine learning toolchains and models with sometimes tens of millions of parameters, making them non-conducive for embedded applications.

Finally, we must mention unsharp masking—a ubiquitous technique for image sharpening. It is known for its speed and simplicity, and is regularly implemented in shading languages and on embedded devices for real-time applications. However, it is typically insufficient for the problem of accurate image deconvolution as will be seen in the Results.

3. THE ALGORITHM

3.1 In One Dimension

Suppose we are given a discrete signal $b[i]$ which has been blurred by a box kernel of radius r. We wish to recover the original signal $f[i]$ from $b[i]$. The algorithm estimates $f[i]$ using iterative approximations, $f_n[i]$. Our FAST-METHOD works in five steps:

1. Start with the average of $b[i+r]$ and $b[i-r]$
2. Subtract the average of $b[i+r+1]$ and $b[i-r-1]$
3. Scale by $2r+1$
4. Add the average of $f_n[i+2r+1]$ and $f_n[i-2r-1]$
5. Use this as the new approximation, $f_{n+1}[i]$

Here, $b[i]$ is used as the first approximation (i.e. $b[i] = f_0[i]$). These steps are illustrated in Figure 2. The algorithm may also be written as a single equation:

$$f_{n+1}[i] = \frac{1}{2}(2r+1)(b[i+r] + b[i-r] - b[i+r+1] - b[i-r-1]) + \frac{1}{2}(f_n[i+2r+1] + f_n[i-2r-1])$$

Assuming the absence of noise and the use of a box blur kernel, this method converges exactly to $f[i]$ in the limit of n. A proof of this is given in Appendix A.

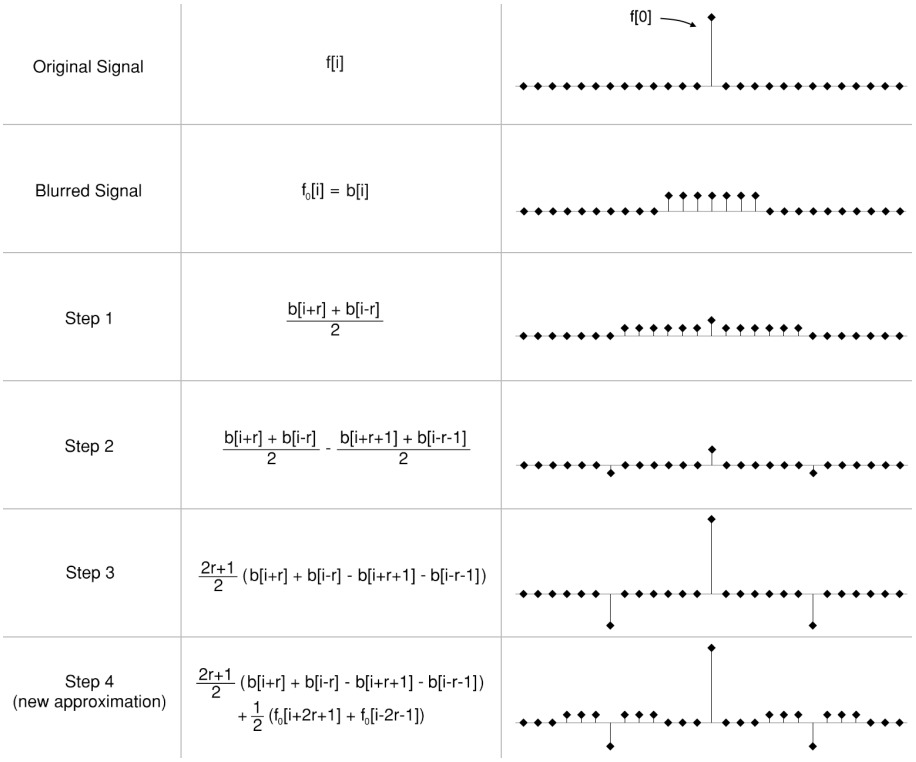

Figure 2. Illustration of steps of the first iteration of FAST-METHOD applied to an example Dirac delta in one dimension. The points represent discrete samples of the signal. The central peak of the original signal is significantly amplified by Step 4.

In pseudocode, the algorithm is implemented as follows:

Algorithm 1: FAST-METHOD in 1D

```
Input: image b, blur radius r
let oldImage = copy of b
repeat
   let newImage = blank array
   for i = 2r+1 to length(b)-2r-1
      newImage[i] = (b[i+r] + b[i-r] - b[i+r+1] - b[i-r-1])/2
      newImage[i] *= 2*r+1
      newImage[i] += (oldImage[i+2r+1] + oldImage[i-2r-1])/2
   end
   oldImage = newImage
until satisfied
return oldImage
```

The above algorithm ignores edge conditions. A practical implementation would clamp the array indices to within the bounds of the signal, effectively extending its borders. The `for` loop may also be parallelized over `i`.

3.2 In Two Dimensions

Suppose we have an image $b(x, y)$ which has been blurred by a disk PSF of radius r. We wish to recover the original signal $f(x, y)$ from $b(x, y)$. The two-dimensional algorithm iteratively approximates $f(x, y)$. The two-dimensional algorithm follows similar steps as in one dimension. For every pixel (x, y), do:

1. Compute the average of all pixels in b that are a distance r from (x, y)
2. Subtract the average of all pixels in b that are a distance of $r+1$ from (x, y)
3. Scale by $0.67/2$, and by the number of pixels averaged in step 1
4. Add the average of all pixels in f_n that are a distance $2r$ from (x, y)
5. Use this as the new approximation, $f_{n+1}(x, y)$

The blurred image is used as the first approximation: $f_0(x, y) = b(x, y)$. The scaling factor of 0.67 on step 3 was determined experimentally. These steps are graphically illustrated in Figure 3. These steps may be equivalently written in equation form:

$$f_{n+1}(x, y) = \frac{1}{2} S_b(r, x, y) - \frac{r}{2(r+1)} S_b(r+1, x, y) + \frac{1}{4\pi r} S_{f_n}(2r, x, y)$$

where

$$S_h(R, x, y) = R \int_0^{2\pi} h(R\cos\theta + x, R\sin\theta + y)\, d\theta.$$

$S_h(R, x, y)$ may be interpreted as the average intensity of every pixel in the specified image h that is exactly distance R from a given pixel (x, y) weighted by the arc length of the integral.

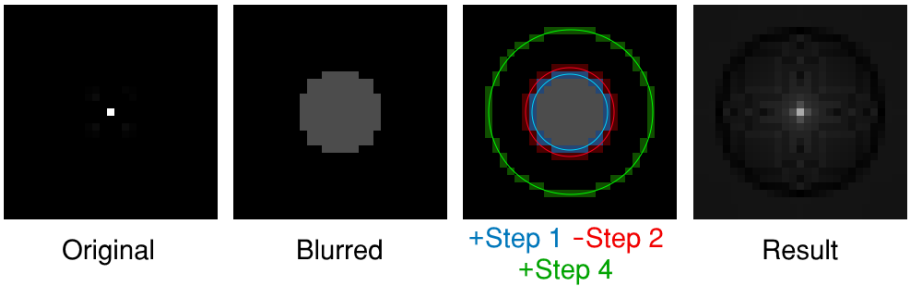

Figure 3. Graphic illustration of the steps of the first iteration of Fast-Method applied to a blurred point. The images are exaggerated for clarity. The result mostly recovers the original image, with some additional ringing effects.

Of course, in a real implementation, the coordinates and integrals must be discretized. A discrete implementation of the 2D Fast-Method is given below:

Algorithm 2: Fast-Method in 2D

```
Input: image b, blur radius r
let oldImage = copy of b
let points1 = computePointsAtRadius(r)
let points2 = computePointsAtRadius(r+1)
let points3 = computePointsAtRadius(2r+1)
repeat
   let newImage = blank 2D array
   for x = 2r+1 to width(b)-2r-1
      for y = 2r+1 to height(b)-2r-1
         let sum1, sum2, sum3 = 0
         for p in points1
            sum1 += b[x + p.x, y + p.y]
         end
         for p in points2
            sum2 += b[x + p.x, y + p.y]
         end
         for p in points3
            sum3 += oldImage[x + p.x, y + p.y]
         end
         sum1 *= 0.67/2
         sum2 *= -0.67/2 * length(points1) / length(points2)
         sum3 /= length(points3)
         newImage[x, y] = sum1 + sum2 + sum3
      end
   end
   oldImage = newImage
until satisfied
return oldImage
```

Here, the `computePointsAtRadius(r)` function returns an array of points that are approximately distance `r` from the origin. For example, `computePointsAtRadius(1)` might return `{(1,0),(-1,0),(0,1),(0,-1)}`. Like the one-dimensional pseudocode, the above algorithm neglects edge conditions. A practical implementation would clamp all coordinates to within the edges of the image. This algorithm is easily parallelized over `x` and `y`, making it suitable for a GPU-accelerated implementation such as in a shader program. Such an implementation is benchmarked in the Results.

Since the algorithm iterates over each pixel once, and sums over approximately $8\pi r$ surrounding pixels in the innermost loops, the computational complexity is $O(n\,r\,k)$ where n is the number of pixels in the image, r is the blur radius, and k is the number of iterations to run the algorithm. (Typically $k=1$ is sufficient.) Additionally, since at any point in the algorithm only two additional copies of the image are needed, the algorithm runs in three times the memory footprint as the input image.

The two-dimensional implementation does have a notable downfall: although it is iterative, it is not guaranteed to converge to the original image $f(x, y)$ even in ideal conditions. It typically produces negligible improvement after the first iteration. Nevertheless, the first iteration alone typically produces useful restoration.

4. RESULTS

We compare our Fast-Method against Richardson-Lucy, Wiener deconvolution, and unsharp masking. All three algorithms are popular in a number of applications. Early results for our method were presented in [22]. Since then, our implementation has improved, and the numbers presented here may not match the previous work.

As a measure of error, we use the normalized root mean square error (NRMSE) which compares an ideal signal a to its degraded counterpart b:

$$\mathrm{NRMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big((a_i - \mathrm{avg}(a)) - (b_i - \mathrm{avg}(b))\big)^2}$$

For color images, each channel is normalized first (by subtracting the average), then added in the summation above. Lower NRMSE typically indicates better deblurring quality.

To measure noise, we use signal-to-noise ratio (SNR):

$$\mathrm{SNR} = \frac{\mathrm{avg}(a)}{\sigma_a}$$

All experiments were performed on an Intel Core i9-10850K at 4.9 GHz, NVIDIA GeForce RTX 3080, and 16 GB of RAM. All CPU-based implementations were parallelized across the 20 available threads.

4.1 Accuracy on audio signals

Here, we test the ability of our algorithm to repair an audio signal degraded by blurring through time. Figure 4 shows the convergence of Fast-Method when applied to flourish.wav[†], a music file with about 17,600,000 samples. The audio was blurred by radius $r = 200$ using the following point spread function:

$$b[i] = \frac{1}{2r+1}\sum_{n=-r}^{r} f[i-n]$$

where f is the original signal and b is blurred. It was then saved to a new WAV file, re-read, and deconvolved using Fast-Method. The blurred audio file sounded muffled upon playback. After 20 iterations of deconvolution, the music sounded indistinguishable from the original. The NRMSE was approximately proportional to $\frac{1}{i^{1/4}}$ at the ith iteration. The deconvolution continued to improve NRMSE up to about 500 iterations, beyond which error began to increase as noise became amplified.

Figure 5 shows a 3000-sample subset of the waveform of the audio file after various number of iterations of Fast-Method. A single iteration of deconvolution significantly recovers the high frequencies present in the original audio. After 100 iterations, the waveform is a very close fit to the original.

† Copyright Microsoft. flourish.wav was converted from flourish.mid which is distributed with many versions of Microsoft Windows.

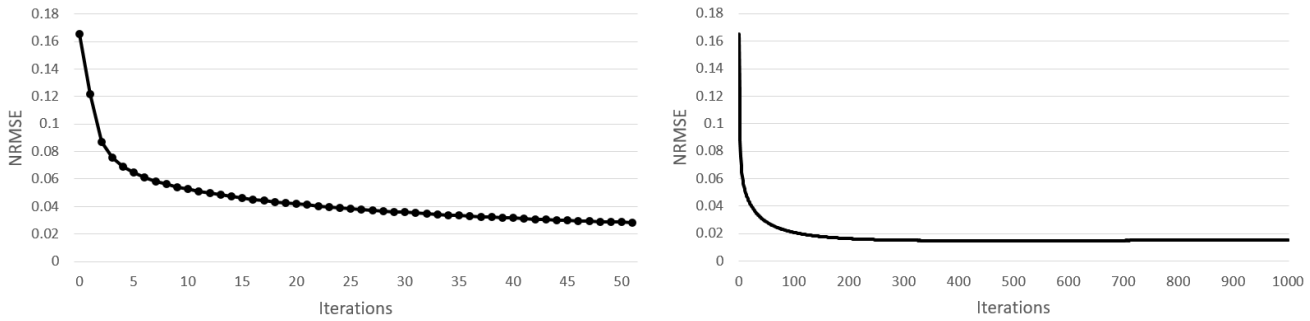

Figure 4. NRMSE vs iterations of Fast-Method applied to audio file flourish.wav. The graph on the left is a magnified view of the first 50 iterations of the graph on the right. The zeroth iteration is the blurred audio with no restoration.

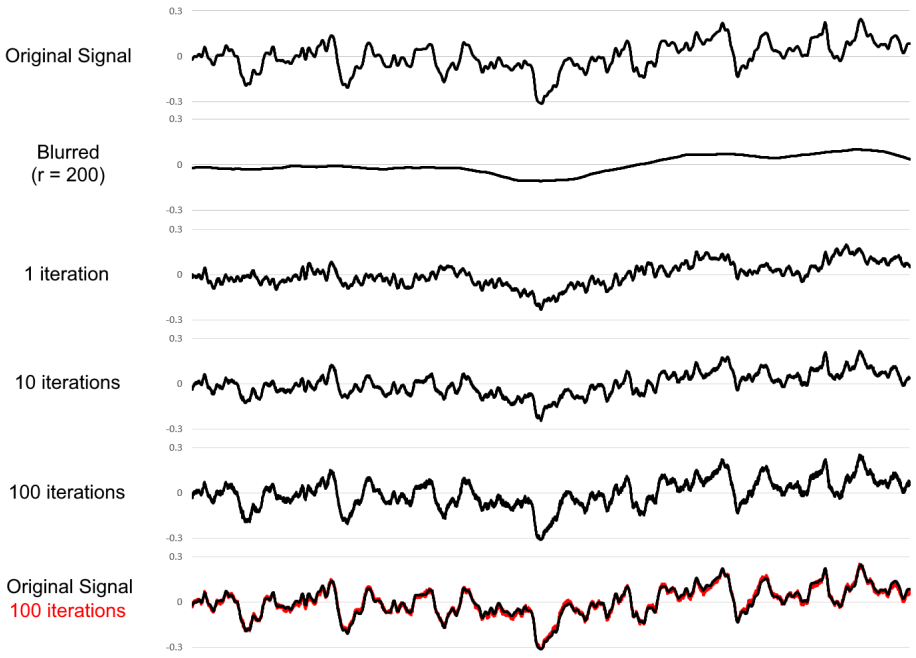

Figure 5. Comparison of different numbers of iterations of Fast-Method on the audio signal from flourish.wav. 3000 samples are shown. The ideal signal is overlaid with the result of 100 iterations in red at the bottom.

4.2 Accuracy and speed on images

For these results, each algorithm, Richardson-Lucy, Wiener, and Fast-Method, has been implemented in multiple languages and parallelized. For Richardson-Lucy, we use a parallelized Java implementation and a parallelized OpenCL implementation for GPU acceleration. For Wiener deconvolution, we use a parallelized C++/OpenCV implementation [3] and a parallelized OpenCV/CUDA implementation for GPU acceleration. For our Fast-Method, we use a parallelized Java implementation, a parallelized OpenCL implementation for GPU acceleration, and an OpenGL shader implementation for real-time processing. Only the Fast-Method could be implemented in OpenGL due to its relative simplicity. Only the implementations which produced the best results are compared.

We start with a comparison of the deblurring quality. Figure 6 shows a comparison of the best quality achieved by each algorithm at various blur radiuses on a 512×512 color image with synthetic disk blur. The parameters for each algorithm, such as iteration count for Richardson-Lucy and SNR for Wiener deconvolution, were tuned to

produce the lowest NRMSE in each case. Richardson-Lucy produced the best quality result while FAST-METHOD produced the worst result. However, FAST-METHOD typically performed orders of magnitude faster than either other method, even when all three algorithms were run on a GPU. All algorithms produced improvement over the unprocessed blurred image. For the FAST-METHOD, a single iteration ($n = 1$) was used. Further iterations gave negligible improvement.

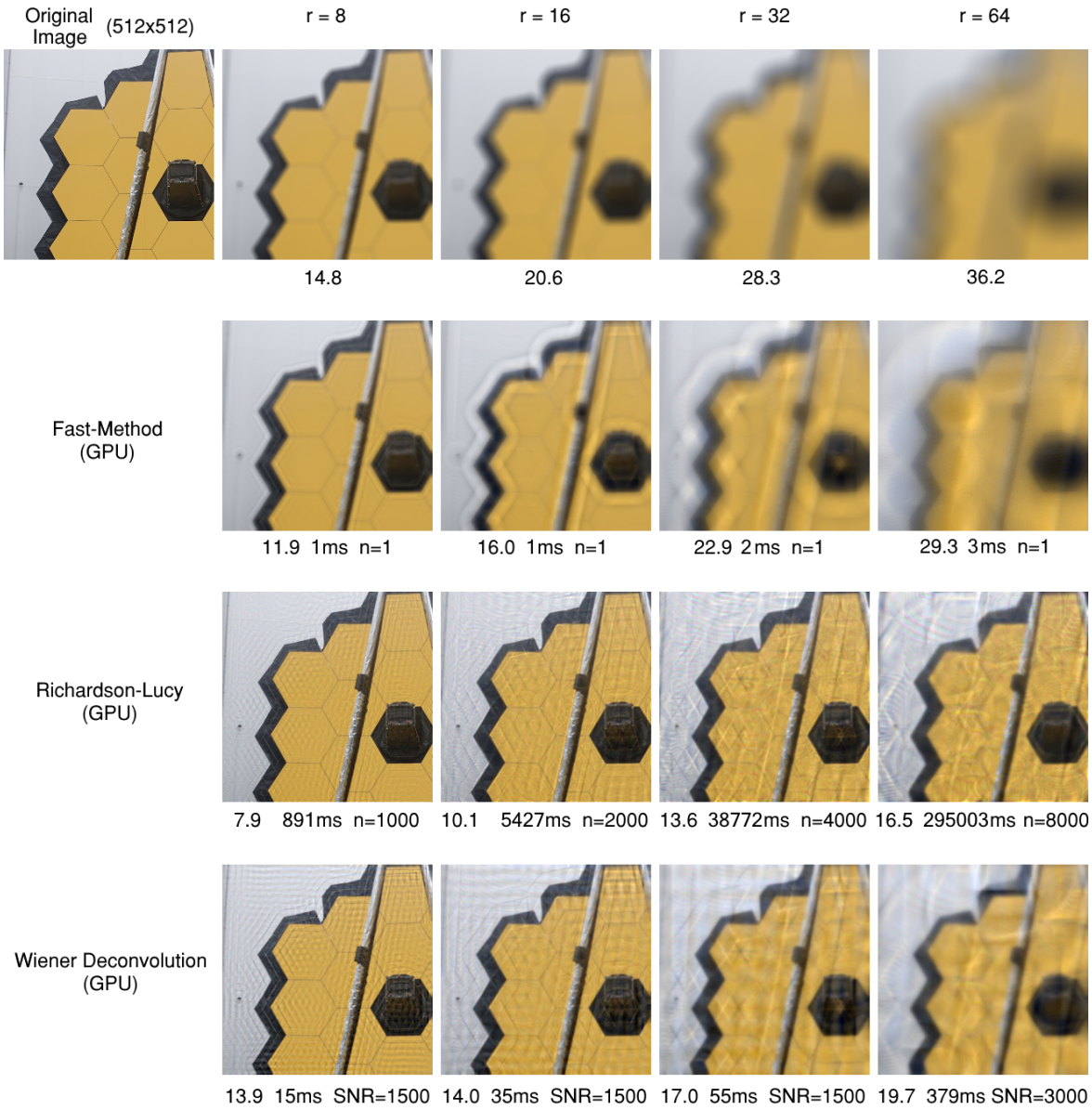

Figure 6. Comparison of deblur quality at various blur radiuses. The top row is the blurred image prior to processing, captioned by their NRMSE. Under each result is NRMSE, time, and parameters for each algorithm.

4.3 Accuracy in the presence of noise

Next, we compare the three methods in the presence of additive Gaussian noise. Figure 7 shows the maximum quality achieved by each method on images degraded first by disk blurring ($r = 6$), then by the addition of

colored noise. As before, the parameters for each algorithm were tuned to produce the best result. Wiener deconvolution and Richardson-Lucy consistently produced lower NRMSE than FAST-METHOD. All algorithms produced improvement over the unprocessed images, except in the case of the FAST-METHOD at SNR=10. This would seem to indicate that the FAST-METHOD does not handle noise as well as the other algorithms.

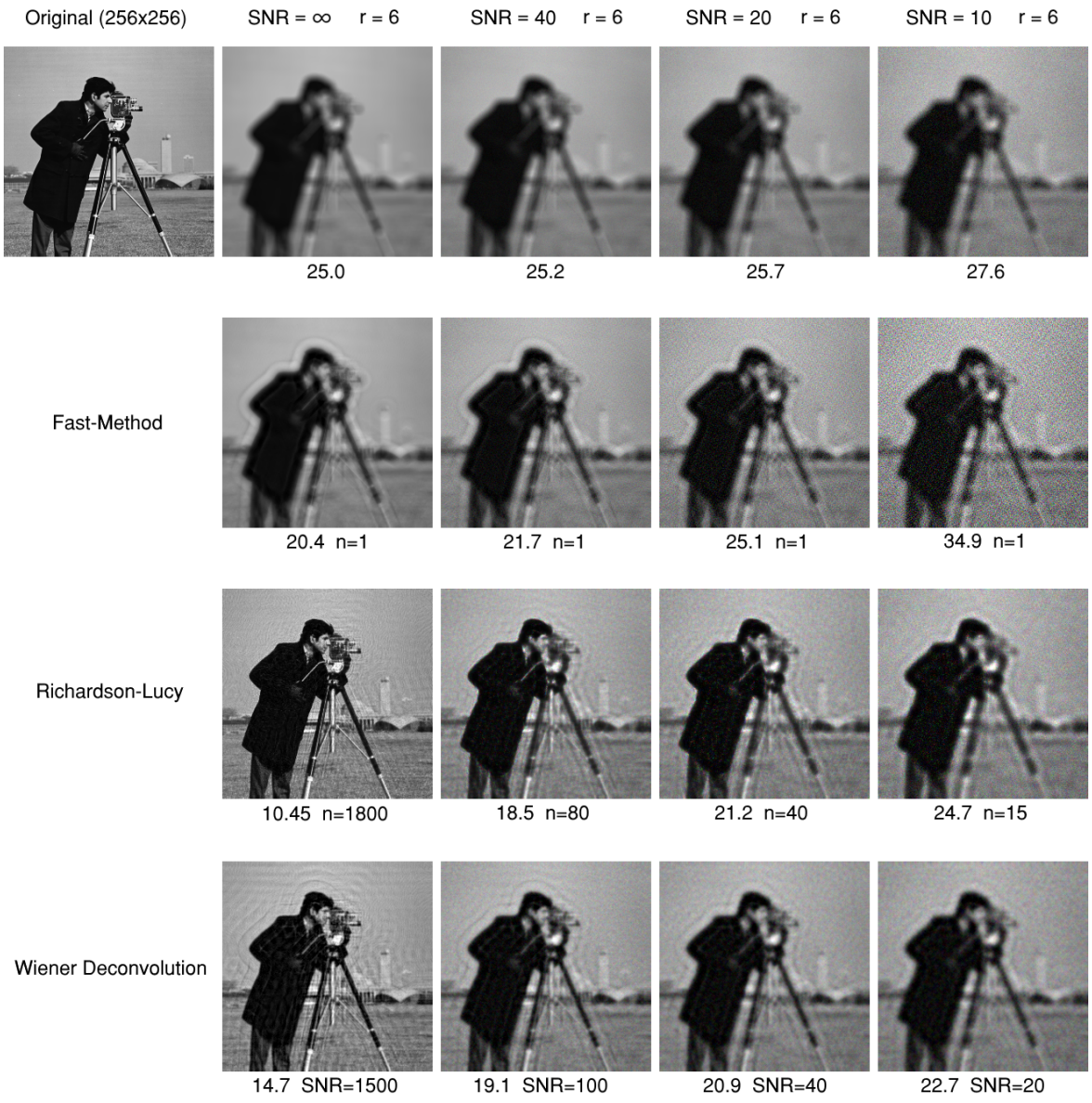

Figure 7. Comparison of deblurring methods in the presence of additive Gaussian noise. The top row is the image prior to deblurring. Under each result is NRMSE and the parameters used for the algorithm. n indicates iterations of each algorithm, while SNR indicates the signal-to-noise ratio configured for Wiener deconvolution. Image courtesy of MIT.

4.4 Accuracy and speed with a real photograph

Next, we compare the quality and performance of each algorithm when applied to a real out-of-focus photograph. Figure 8 shows the result of deblurring a photograph of text with each of the four algorithms. Text was printed

on a sheet of paper and photographed with a Sony DSC-RX100M4 with f-stop f/2.8, 1/160s exposure, and a focal length of 23mm—intentionally out of focus, then saved as a compressed JPEG with resolution 5472×3080. The true radius of the blur was then estimated by guess and check—deblurring with various radiuses until the best setting was found. All the algorithms except unsharp mask restored the text to a legible degree. FAST-METHOD and unsharp masking performed 69× faster than Wiener deconvolution and 29,762× faster than Richardson-Lucy, while Wiener deconvolution produced the best result visually. In this case, the accuracy of Richardson-Lucy was limited by its lengthy runtime (about 9 minutes).

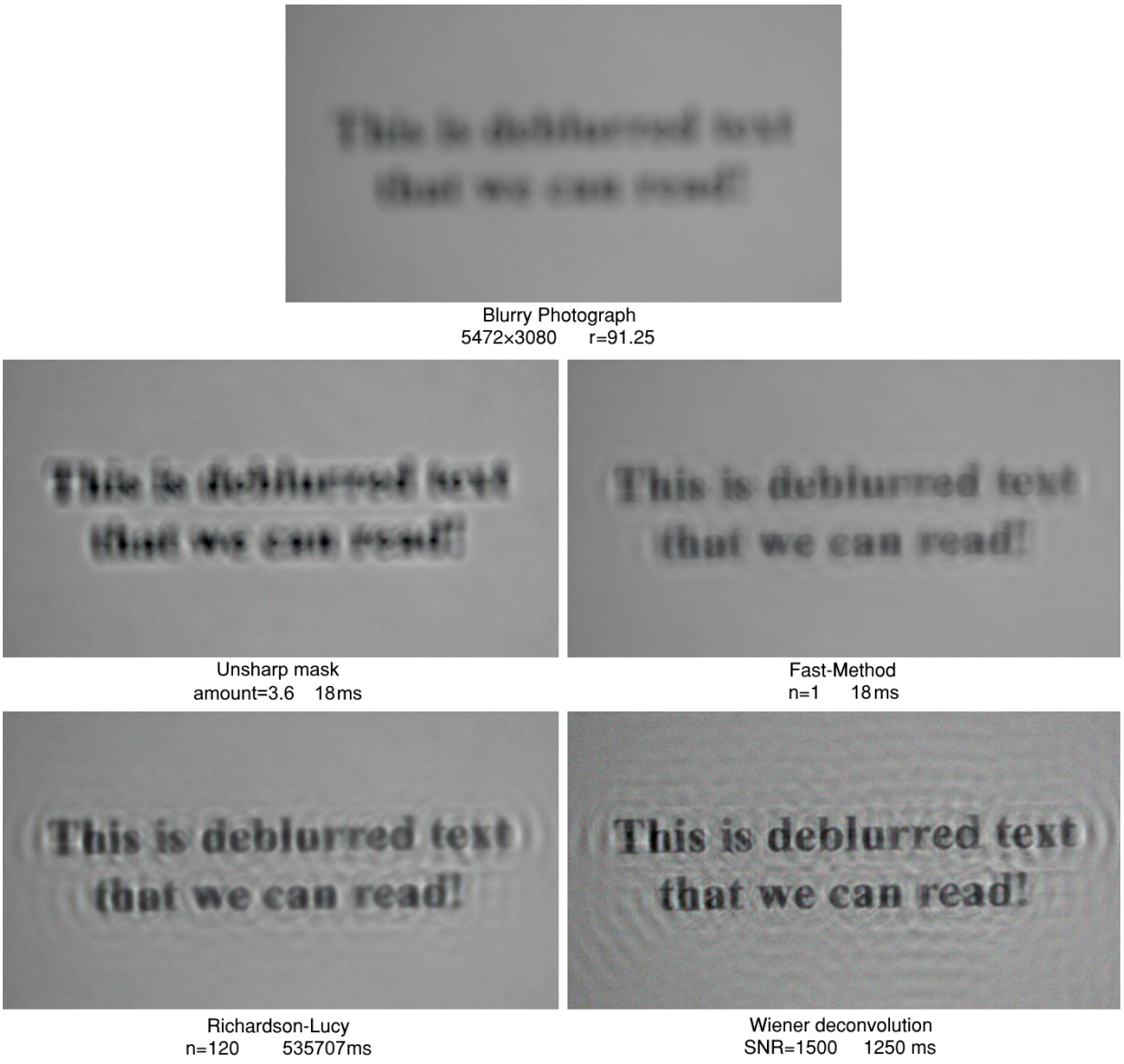

Figure 8. Comparison of deblurring methods on an out-of-focus photograph.

Despite the poor performance of the FAST-METHOD on noisy images in the previous section, the top-right image in Figure 8 shows clear improvement over the blurry image, even in the presence of significant noise. This noise

can be seen in Figure 9, which shows a close-up view of the original image used in Figure 8.

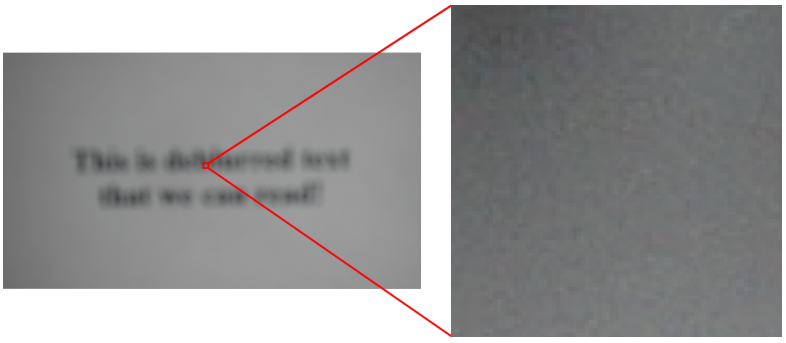

Figure 9. Magnified view of a small portion of the original photograph used in Figure 8. Significant noise and JPEG compression artifacts are visible.

4.5 Comparison against unsharp masking

Next, we perform a more detailed comparison between Fast-Method and unsharp masking. We test implementations of unsharp masking in Java, OpenCL, and OpenGL. For comparisons, we use real high-resolution (5496×3672px) photographs captured by a Sony DSC-RX100M4 intentionally defocused by varying amounts, including an in-focus image for reference. Figure 10 shows the result of applying the unsharp mask and Fast-Method to each photograph along with the NRMSE and computation time of each implementation of each algorithm. In each case, the intensity of the unsharp mask is adjusted to minimize the NRMSE. NRMSE is calculated by first registering, then subtracting each image from the reference in-focus image.

At larger blur radiuses, Fast-Method produces better results, revealing more sharp edges and recovering more text than in the unsharp masked images while requiring less execution time. Only at a blur radius of r=5.2 does the unsharp mask produce a lower NRMSE. Both methods require 22ms when executed in OpenGL. This indicates that both algorithms' execution time is negligible compared to the overhead of invoking the computation.

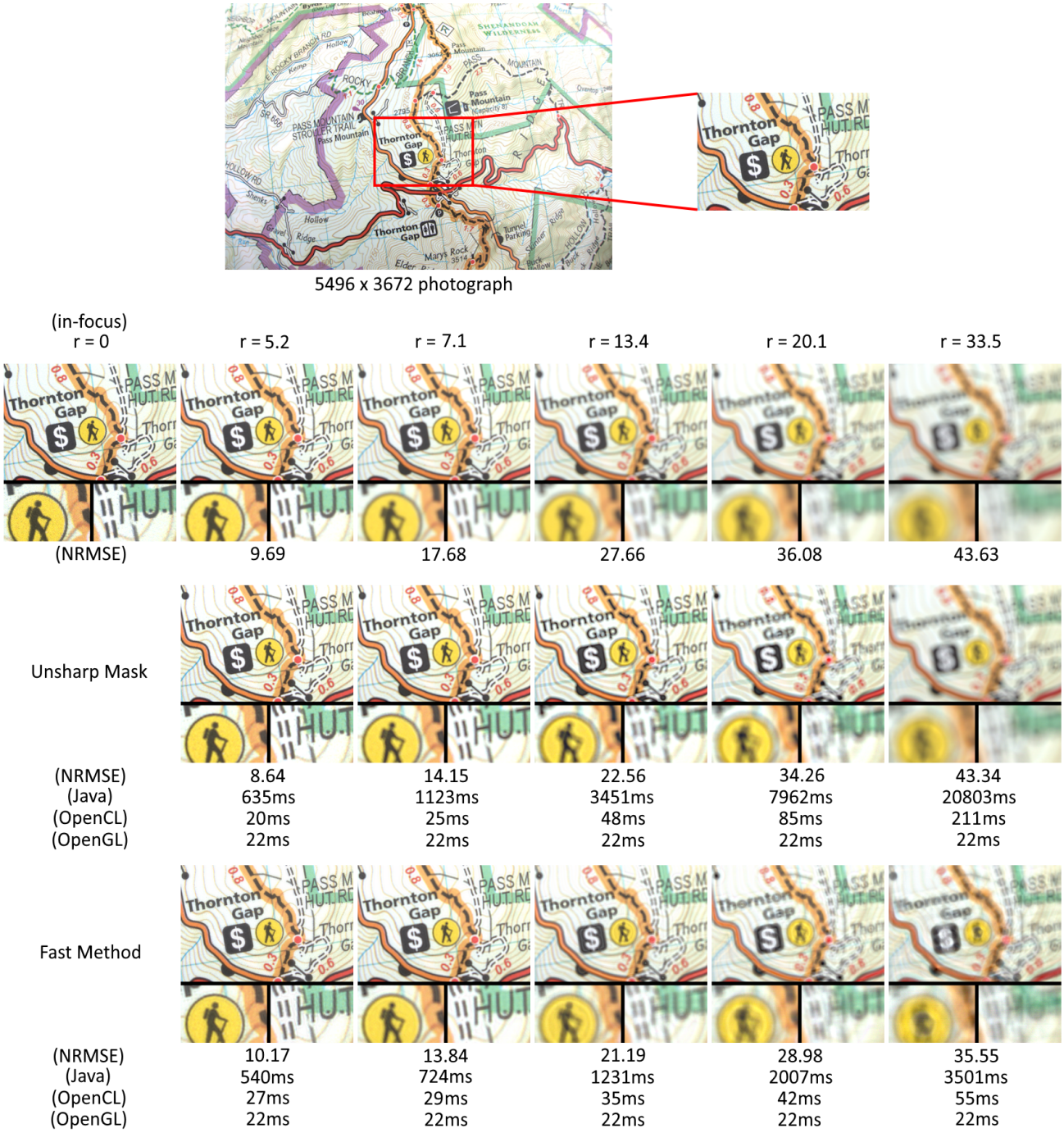

Figure 10. Comparison of deblurring time and quality of various implementations of Fast-Method and the unsharp mask. Each image is cropped from its corresponding full-size image, like that shown at the top. Below each image is NRMSE and execution time if applicable. Top row: original image captured by a camera out-of-focus by the specified blur radiuses. Second row: result of unsharp masking. Third row: result of Fast-Method.

4.6 Time to constant NRMSE

Next, we compare the performance of each algorithm by measuring the time required to achieve a constant accuracy. Table 1 shows the performance of each implementation for each algorithm on a 1920×1080 color

image blurred by $r = 16$. Each algorithm's parameters were adjusted to produce the same quality result (with NRMSE≈12.83), excluding the unsharp mask which could not attain the desired accuracy. The OpenGL implementations of Fast-Method and unsharp masking were by far the fastest. It is worth noting that the CPU implementation of Wiener deconvolution performed 1.35× faster than the CPU implementation of Fast-Method, yet the OpenGL implementation of Fast-Method performed 28× faster than our fastest implementation of Wiener deconvolution.

Table 1. Comparison of performance of different implementations of each algorithm where NRMSE is held equal. The test image is 1920×1080 color, blurred by $r = 16$. Blank cells indicate that a particular implementation was not obtained.

Implementation	Richardson-Lucy (n=31)	Wiener deconvolution (SNR=1500)	Fast-Method (n=1)	Unsharp Mask
Java (CPU)	39416 ms	-	156 ms	512 ms
OpenCL (GPU)	1685 ms	-	10 ms	16 ms
OpenGL (GPU)	-	-	3 ms	3 ms
C++/OpenCV (CPU)	-	115 ms	-	-
C++/OpenCV (GPU)	-	84 ms	-	-

4.7 Video processing performance

Finally, we analyze the performance of Fast-Method in the application of real-time video processing. Tables 2 and 3 show the performance of Fast-Method when deblurring video. To achieve real-time speeds, the OpenGL implementation was used. The video file was 1080p 3-channel color at 10,022kbps in MP4 format. The 4K/UHD video was 15,500kbps. Each frame was read off a solid-state drive, decoded, deblurred, and displayed on the screen in sequence. The process was pipelined to allow for simultaneous decoding and deblurring. The frame rates reported were averaged over the entire length of the video. Our implementation was able to process the 1080p video at up to 122 frames per second—much faster than real-time, and the 4K/UHD video consistently at 32 fps. Even as the blur radius was increased, the execution time of Fast-Method only exceeded the I/O overhead at a 60px radius blur for 1080p video. For 4K/UHD video, the frame rate was not limited by the execution of Fast-Method and the GPU's capability was never saturated. Figure 11 shows an example video.

Table 2. Video processing rate of Fast-Method at various blur radiuses on 1080p video. "Total frame time" is the time between frames as they are displayed on the screen. "Fast-Method time" is only the time taken by the execution of Fast-Method while processing each video frame, excluding disk I/O and decoding time. Frame rate excluding disk I/O is calculated from "Fast-Method time". GPU usage was reported by Argus Monitor‡.

Blur radius	Frame rate	Frame rate (excluding disk I/O)	Total frame time	Fast-Method time	GPU Usage
r=9	122 fps	364 fps	8.2 ms	2.8 ms	39%
r=21	122 fps	364 fps	8.2 ms	2.8 ms	59%
r=40	122 fps	364 fps	8.2 ms	2.8 ms	96%
r=60	89 fps	91 fps	11.2 ms	11.0 ms	97%

Table 3. Similar to Table 2 above, but for 4K/UHD video.

Blur radius	Frame rate	Frame rate (excluding disk I/O)	Total frame time	Fast-Method time	GPU Usage
r=9	32 fps	99 fps	30.8 ms	10.1 ms	30%
r=21	32 fps	101 fps	30.8 ms	9.9 ms	33%
r=40	33 fps	99 fps	30.6 ms	10.1 ms	38%
r=60	32 fps	98 fps	30.7 ms	10.2 ms	44%

‡ Argus Monitor, Copyright © Argotronic GmbH

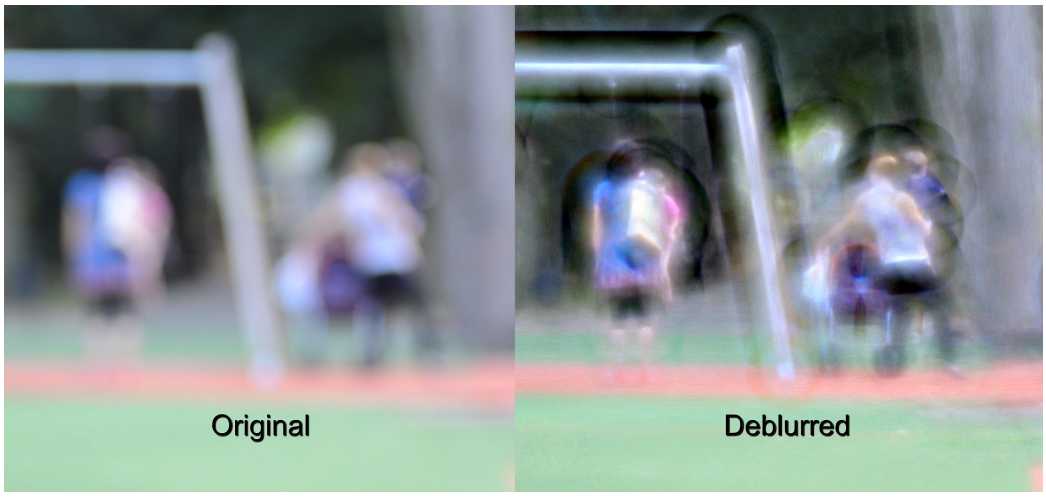

Figure 11. Example of video processing performance on freely available stock footage. Left: original video downloaded from videvo.net. Right: deblurred video processed by our method at 122 fps. The blur radius was estimated to be 37 pixels. Video is copyright Freepik, used with permission. See supporting material for the video file.

5. APPLICATIONS

The proposed FAST-METHOD may find use wherever speed is a priority over quality. Here, we list a few applications.

(i) Real-time video processing: Given the performance metrics in Table 1, only FAST-METHOD is a candidate for real time video processing. For real-time imaging systems such as microscopes, an operator may need to make timely decisions based on a dynamic specimen. Thus, it is best if all image processing is performed in real-time.

(ii) Real-time audio processing: Deblurring imagery is mathematically akin to de-muffling audio, which can improve clarity. Our algorithm may be used to perform de-muffling in real-time, which may find uses in audio conferencing or recovering attenuated recordings.

(iii) Starting approximation for other algorithms: Many iterative deconvolution algorithms, including Richardson-Lucy, require a starting approximation. Often, the blurred image is simply used as a starting point. A better starting approximation can accelerate the process. Our FAST-METHOD may provide a good approximation to jump-start lengthier iterative methods. Additionally, deep-learning based methods such as [20] can benefit from additional images containing useful spatial information. We can imagine training a model to use the output of our method to augment its own image restoration.

(iv) Estimation of blur radius: The general problem of estimating a point spread function is called blind deconvolution. If we are restricted to only disk-like blurs, then the FAST-METHOD offers a convenient way to visually estimate the radius of such a disk blur. It is sufficiently quick that a user may interactively adjust parameters until the image comes into focus. Indeed, this convenience was used extensively in this research.

6. CONCLUSION

We have proposed a deconvolution algorithm which performs significantly faster than most other methods tested. It is faster than unsharp masking while recovering more image detail. Thanks to the simplicity and parallelizability of the algorithm, it is relatively easy to implement in a variety of languages including C, Java, OpenCL, and OpenGL. Using the OpenGL implementation, we were able to deblur 1080p video at up to 122 fps and 4K/UHD video at 32 fps. Although it performed very quickly, it produced less accurate results than other time-consuming image deblurring methods tested. However, for one-dimensional signals, we mathematically proved and experimentally demonstrated that our method converges to the ideal signal in the absence of noise and can be used to recover degraded audio signals.

Source code is available at https://github.com/dwilliams1114/fast-deblur

7. FUTURE WORK

- Although we've only discussed deblurring images degraded by disk-like point spread functions, the proposed algorithm could be modified to repair images blurred by other simple point spread functions with clearly defined edges, such as ellipses or boxcar functions.

- Another common case we haven't explored is the repair of motion blurred images. Assuming a linear blur, in principle this isn't too difficult. By applying the one-dimensional FAST-METHOD to a two-dimensional image in the direction of the blur, we would expect a decent result like that achieved in one dimension.

- We demonstrated the usefulness of our method in one and two dimensions. However, this method could be extended three dimensions using the same logic used in the one- and two-dimensional implementations.

- Our method does not achieve state-of-the-art quality. However, it does provide a quick approximation that may be used to guide or accelerate a more accurate deep-learning-based defocus deblurring algorithm.

BIOGRAPHY

Daniel Williams received his BS in computer science from University of Houston at Clear Lake in 2020. He enjoys the pursuit of better and faster algorithms in his leisure. His research interests include image processing, automated design of algorithms, novel machine learning, and theoretical models for computation. Currently, he works at Helios Solutions supporting Intuitive Machines developing spaceflight software.

DISCLOSURES

No conflicts of interest exist in this research.

CODE, DATA, AND MATERIALS AVAILABILITY

Source code is available at https://github.com/dwilliams1114/fast-deblur. A subset of test images are included in the repo. The full set of test images and video files will be made available upon request.

REFERENCES

[1] Richardson, W. H., Bayesian-Based Iterative Method of Image Restoration. J. Opt. Soc. Amer. 62(1), 55-59 (1972).

[2] Lucy, L. B., An iterative technique for the rectification of observed distributions. Astronom. J. 79(6), 745-754 (1974).

[3] Vladislav, K., Out-of-focus Deblur Filter. OpenCV, 17 July 2018, https://docs.opencv.org/4.x/de/d3c/tutorial_out_of_focus_deblur_filter.html (27 July 2023).

[4] Kaihao Zhang, Wenqi Ren, Wenhan Luo, Wei-Sheng Lai, Bjorn Stenger, Ming-Hsuan Yang, and Hongdong Li. Deep image deblurring: A survey. International Journal of Computer Vision, 130(9):2103–2130, 2022

[5] El-Henawy I, Amin A, Kareem Ahmed HA. 2014. A comparative study on image deblurring techniques. Int. J. Adv. Comput. Sci. Technol. 3:1–8

[6] Berisha, S. & Nagy, J. G. Iterative methods for image restoration. In Academic Press Library in Signal Processing, vol. 4, 193–247 (Elsevier, 2014).

[7] Fliegel, K.; Janout, P.; Bednář, J.; Krasula, L.; Vítek, S.; Švihlík, J.; Páta, P. Performance Evaluation of Image Deconvolution Techniques in Space-Variant Astronomical Imaging Systems with Nonlinearities. In SPIE Proceedings Vol. 9599: Applications of Digital Image Processing XXXVIII; Tescher, A.G., Ed.; SPIE: Washington, DC, USA, 2015; p. 959927.

[8] D. Krishnan and R. Fergus, Fast image deconvolution using hyper-Laplacian priors, in Proc. Adv. Neural Inf. Process. Syst., 2009, pp. 1033–1041

[9] J. Klosowski, S. Krishnan, Real-time image deconvolution on the GPU, in: SPIE Conference: Parallel Processing for Imaging Applications, Jan. 2011.

[10] Xu, L., Tao, X., Jia, J.: Inverse kernels for fast spatial deconvolution, in: European Conference on Computer Vision (2014)

[11] Zhao B., Mertz J. Resolution enhancement with deblurring by pixel reassignment Advanced Photonics, 5 (6) (2023), Article 066004

[12] Hyun Kim, T., Mu Lee, K., Scholkopf, B., Hirsch, M.: Online video deblurring via dynamic temporal blending network. In: IEEE International Conference on Computer Vision (2017)

[13] Tsai, F.-J.; Peng, Y.-T.; Lin, Y.-Y.; Tsai, C.-C.; Lin, C.-W. Stripformer: Strip Transformer for Fast Image Deblurring. arXiv 2022.

[14] Orest Kupyn, Tetiana Martyniuk, Junru Wu, and Zhangyang Wang. DeblurGAN-v2: Deblurring (orders-ofmagnitude) faster and better. In ICCV, 2019.

[15] Hu, X., Ren, W., Yu, K., Zhang, K., Cao, X., Liu, W., Menze, B.: Pyramid architecture search for real-time image deblurring. In: Proc. Int'l Conf. Computer Vision (2021)

[16] Yuan Yuan, Wei Su, and Dandan Ma. Efficient dynamic scene deblurring using spatially variant deconvolution network with optical flow guided training. In IEEE Conf. Comput. Vis. Pattern Recog., pages 3555–3564, 2020.

[17] Tao, X., Gao, H., Shen, X., Wang, J., Jia, J.: Scale-recurrent network for deep image deblurring. In: IEEE Conference on Computer Vision and Pattern Recognition (2018)

[18] Nah, S., Hyun Kim, T., Mu Lee, K.: Deep multi-scale convolutional neural network for dynamic scene deblurring. In: IEEE Conference on Computer Vision and Pattern Recognition (2017)

[19] Zhang, K., Luo, W., Zhong, Y., Stenger, B., Ma, L., Liu, W., Li, H.: Deblurring by realistic blurring. In: IEEE Conference on Computer Vision and Pattern Recognition (2020)

[20] Abuolaim, A., Brown, M.S.: Defocus deblurring using dual-pixel data. In: European Conference on Computer Vision (2020)

[21] Junyong Lee, Hyeongseok Son, Jaesung Rim, Sunghyun Cho, and Seungyong Lee. Iterative filter adaptive network for single image defocus deblurring. In CVPR, 2021.

[22] Williams, D: A fast, simple, and parallelizable deconvolution algorithm for real-time applications, Proc. SPIE 12674, Applications of Digital Image Processing XLVI, 126740B (2023)

APPENDIX

A Proof of convergence in one dimension

In this appendix, we prove that the proposed algorithm converges exactly to the original un-blurred signal when applied iteratively to discretely sampled one-dimensional data. To simplify analysis, we will make a few assumptions:

1. Assume the proposed algorithm is being applied to an infinite signal. It is well known that exact deconvolution of finite signal near its edges is impossible simply because the blurred signal is not known beyond the edges. In practice, the blur radius is typically much smaller than the length of the signal, so the error due to edge effects is minute.

2. Assume the blurred signal is noiseless for simplicity.

3. Assume the blurred signal is Lebesgue integrable. For all practical purposes, this means that the signal generally decreases to zero toward $\pm\infty$ and has a finite sum. This is not far fetched, as real-world signals are typically finite and can be thought of as being padded with infinitely many zeros.

4. Assume the original signal (and thus blurred signal) is non-negative everywhere. If needed, a constant may be added to the entire signal to ensure it is non-negative.

5. Assume the blur radius r is a positive integer.

Here, we give a precise description of the proposed algorithm. Let i represent a discrete index of the signal. In one dimension, the disk point spread function is defined as

$$g(i) = \frac{1}{2r+1} \begin{cases} 1, & |i| \le r, \\ 0, & |i| > r \end{cases}$$

and so the blurred sample $b(i)$ at index i is

$$b(i) = (f * g)(i) = \sum_{n=-\infty}^{\infty} f(i-n)\, g(n) = \frac{1}{2r+1} \sum_{n=-r}^{r} f(i-n). \tag{1}$$

This is equivalent to averaging $2r+1$ adjacent samples. The procedure of the proposed algorithm is defined as

$$\begin{aligned} f_{n+1} &= b * A + f_n * B \\ f_0 &= b \end{aligned} \tag{2}$$

where f_n is the nth approximation of the original signal f, and A and B are defined in one-dimensional space as

$$\begin{aligned} A(i) &= \frac{2r+1}{2}(\delta[i-r] + \delta[i+r] - \delta[i-r-1] - \delta[i+r+1]) \\ B(i) &= \frac{1}{2}(\delta[i-2r-1] + \delta[i+2r+1]) \end{aligned}$$

where δ is the discrete unit sample function (such that $\delta[0] = 1$). In the two-dimensional case, different definitions of A and B would be used, but the iterative procedure would be the same.

Theorem 1. $\lim_{n\to\infty}(f_n) = f$

In other words, the proposed algorithm converges exactly (in one-dimension) to the original undegraded signal, assuming the blurred signal is used as the first approximation and the blurred signal is noiseless.

Proof.

The proof works in the frequency domain. We will define the discrete-time Fourier transform (DTFT) of example function $h(n)$ as

$$\hat{h}(\omega) = \mathcal{F}(h(n)) = \sum_{n=-\infty}^{\infty} h(n) e^{-i\omega n}.$$

We will omit writing function's arguments throughout much of the proof for conciseness (i.e. $\hat{h}$ instead of $\hat{h}(\omega)$). We start with the iterative procedure for the proposed algorithm from (2).

$$f_{n+1} = b * A + f_n * B$$

Applying the DTFT gives

$$\hat{f}_{n+1} = \widehat{b * A} + \widehat{f_n * B}$$

and by the convolution theorem,

$$\hat{f}_{n+1} = \hat{b}\hat{A} + \hat{f}_n \hat{B}.$$

We can solve for f_n explicitly using the recurrence relation

$$\begin{aligned} a_{n+1} &= C + a_n D \text{ and } a_0 = E \\ \therefore \quad a_n &= ED^n + \frac{C(D^n - 1)}{D - 1} \end{aligned}$$

thus

$$\begin{aligned} \underbrace{\hat{f}_{n+1}}_{a_{n+1}} &= \underbrace{\hat{b}\hat{A}}_{C} + \underbrace{\hat{f}_n \hat{B}}_{a_n D} \text{ and } \hat{f}_0 = \underbrace{\hat{b}}_{E} \\ \therefore \quad \hat{f}_n &= \hat{b}\hat{B}^n + \frac{\hat{b}\hat{A}(\hat{B}^n - 1)}{\hat{B} - 1}. \end{aligned}$$

Next we substitute the definition of b as $b(i) = (f * g)(i)$, and apply the convolution theorem again.

$$\begin{aligned} \hat{f}_n &= \widehat{(f * g)}\hat{B}^n + \frac{\widehat{(f * g)}\hat{A}(\hat{B}^n - 1)}{\hat{B} - 1} \\ &= \hat{f}\hat{g}\hat{B}^n + \frac{\hat{f}\hat{g}\hat{A}(\hat{B}^n - 1)}{\hat{B} - 1} \\ &= \hat{f}\left(\hat{g}\hat{B}^n + \frac{\hat{g}\hat{A}(\hat{B}^n - 1)}{\hat{B} - 1} \right). \end{aligned} \tag{3}$$

Taking the limit in iteration n of both sides gives

$$\begin{aligned} \lim_{n\to\infty} (\hat{f}_n) &= \lim_{n\to\infty} \left(\hat{f}\left(\hat{g}\hat{B}^n + \frac{\hat{g}\hat{A}(\hat{B}^n - 1)}{\hat{B} - 1} \right) \right) \\ &= \hat{f}\left(\lim_{n\to\infty} (\hat{g}\hat{B}^n) + \lim_{n\to\infty} \left(\frac{\hat{g}\hat{A}(\hat{B}^n - 1)}{\hat{B} - 1} \right) \right) \\ &= \hat{f}\left(\cancelto{0}{\lim_{n\to\infty} (\hat{g}\hat{B}^n)} + \lim_{n\to\infty} \left(\frac{\hat{g}\hat{A}\cancelto{-1}{(\hat{B}^n - 1)}}{\hat{B} - 1} \right) \right) \\ &= -\hat{f}\frac{\hat{g}\hat{A}}{\hat{B} - 1} \end{aligned} \tag{4}$$

assuming $|\hat{B}|<1$. We will later validate this assumption.

Equation (4) is particularly beautiful because it has no dependence on dimension, choice of Fourier transform, or choice of point spread function g. In theory, any solution for $\hat{A}$ and $\hat{B}$ which satisfies $\frac{\hat{g}\hat{A}}{\hat{B}-1}=-1$ would yield a new deconvolution algorithm. The rest of this proof will apply only to the one-dimensional case.

Next, we compute the DTFT of g, A, and B, then substitute them into equation (4).

$$\begin{aligned}
\hat{g}(\omega) &= \mathcal{F}\Big(\tfrac{1}{2r+1}\begin{cases} 1, & |i|\le r, \\ 0, & |i|>r \end{cases}\Big) \\
&= \tfrac{1}{2r+1}\mathcal{F}\Big(\begin{cases} 1, & |i|\le r, \\ 0, & |i|>r \end{cases}\Big) \\
&= \tfrac{1}{2r+1}\tfrac{\sin(r\omega+\omega/2)}{\sin(\omega/2)}
\end{aligned}$$

$$\begin{aligned}
\hat{A}(\omega) &= \mathcal{F}\big(\tfrac{2r+1}{2}(\delta[i-r]+\delta[i+r]-\delta[i-r-1]-\delta[i+r+1])\big) \\
&= \tfrac{2r+1}{2}(\mathcal{F}(\delta[i-r])+\mathcal{F}(\delta[i+r])-\mathcal{F}(\delta[i-r-1])-\mathcal{F}(\delta[i+r+1])) \\
&= \tfrac{2r+1}{2}(e^{-ir\omega}+e^{ir\omega}-e^{-i(r+1)\omega}-e^{i(r+1)\omega}) \\
&= (2r+1)(\cos(r\omega)-\cos(r\omega+\omega))
\end{aligned}$$

$$\begin{aligned}
\hat{B}(\omega) &= \mathcal{F}\big(\tfrac{1}{2}(\delta[i-2r-1]+\delta[i+2r+1])\big) \\
&= \tfrac{1}{2}\mathcal{F}(\delta[i-2r-1])+\tfrac{1}{2}\mathcal{F}(\delta[i+2r+1]) \\
&= \tfrac{1}{2}e^{-i(2r+1)\omega}+\tfrac{1}{2}e^{i(2r+1)\omega} \\
&= \cos(2r\omega+\omega)
\end{aligned}$$

Notice that since $\hat{B}(\omega)=\cos(2r\omega+\omega)$, our assumption that $|\hat{B}|<1$ is met almost everywhere. Since there are only countably many locations where the assumption is not met (where $\omega=\frac{\pi n}{2r+1}$), they will contribute nothing to the numerical value of the integral (for the inverse-DTFT) later, and we can safely ignore this issue. Substituting each of the above into (4),

$$\begin{aligned}
\lim_{n\to\infty}(\hat{f}_n) &= -\hat{f}\,\frac{\hat{g}\hat{A}}{\hat{B}-1} && \text{(from (4))} \\
&= -\hat{f}\,\frac{\frac{1}{2r+1}\frac{\sin(\omega(r+1/2))}{\sin(\omega/2)}(2r+1)(\cos(r\omega)-\cos((r+1)w))}{\cos(2\omega r+\omega)-1} && \text{(substitution)} \\
&= -\hat{f}\,\frac{\sin(r\omega+\omega/2)(\cos(r\omega)-\cos(r\omega+\omega))}{\sin(\omega/2)(\cos(2r\omega+\omega)-1)} && \text{(algebra)} \\
&= \hat{f}\,\frac{\sin(r\omega+\omega/2)(\cos(r\omega)-\cos(r\omega+\omega))}{2\sin(\omega/2)\sin^2(r\omega+\omega/2)} && \text{(power reduction)} \\
&= \hat{f}\,\frac{\cos(r\omega)-\cos(r\omega+\omega)}{2\sin(\omega/2)\sin(r\omega+\omega/2)} && \text{(cancellation)} \\
&= \hat{f}\,\frac{-2\sin\big(\frac{r\omega-r\omega-\omega}{2}\big)\sin\big(\frac{r\omega+r\omega+\omega}{2}\big)}{2\sin(\omega/2)\sin(r\omega+\omega/2)} && \text{(sum-to-product)} \\
&= \hat{f}\,\frac{-\sin(-\omega/2)\sin(r\omega+\omega/2)}{\sin(\omega/2)\sin(r\omega+\omega/2)} && \text{(algebra)} \\
&= \hat{f}\,\frac{\sin(\omega/2)\sin(r\omega+\omega/2)}{\sin(\omega/2)\sin(r\omega+\omega/2)} && \text{(symmetry of sine)} \\
&= \hat{f} && (5)
\end{aligned}$$

and taking the inverse DTFT of both sides gives

$$\begin{aligned}\mathcal{F}^{-1}\Big(\lim_{n\to\infty}(\hat{f}_n)\Big) &= \mathcal{F}^{-1}(\hat{f}) \\ &= f\end{aligned}$$

which may be written explicitly as

$$\begin{aligned}\frac{1}{2\pi}\int_{-\pi}^{\pi}\lim_{n\to\infty}(\hat{f}_n)\,e^{i\omega t}\,d\omega &= f \\ \frac{1}{2\pi}\int_{-\pi}^{\pi}\lim_{n\to\infty}(\hat{f}_n\,e^{i\omega t})\,d\omega &= f. \end{aligned} \tag{6}$$

To exchange the integral and the limit, we check the following conditions and apply the dominated convergence theorem:

1. Are $\hat{f}_n\,e^{i\omega t}$ all integrable over the range $\omega\in[-\pi,\pi]$? We assumed that f is integrable over t, and since f_n are generated by additions and convolutions with integrable functions, f_n must be integrable, and so $\mathcal{F}(f_n)=\hat{f}_n$ is integrable over $\omega\in[-\pi,\pi]$, and since $|\hat{f}_n\,e^{i\omega t}|=|\hat{f}_n|$, $\hat{f}_n\,e^{i\omega t}$ is also integrable over $\omega\in[-\pi,\pi]$.
2. Are $|\hat{f}_n\,e^{i\omega t}|$ bounded by an integrable function almost everywhere in $\omega\in[-\pi,\pi]$? (Justification follows)

To determine whether $|\hat{f}_n\,e^{i\omega t}|$ are bounded, we first determine whether $\left|\frac{\hat{f}_n}{\hat{f}}\right|$ are bounded. Starting from (3), we can divide by $\hat{f}$,

$$\begin{aligned}\hat{f}_n &= \hat{f}\left(\hat{g}\hat{B}^n+\frac{\hat{g}\hat{A}(\hat{B}^n-1)}{\hat{B}-1}\right) \\ \frac{\hat{f}_n}{\hat{f}} &= \hat{g}\hat{B}^n+\frac{\hat{g}\hat{A}(\hat{B}^n-1)}{\hat{B}-1} \\ \frac{\hat{f}_n}{\hat{f}} &= \hat{g}\hat{B}^n+\frac{\hat{g}\hat{A}}{\hat{B}-1}(\hat{B}^n-1)\end{aligned} \tag{7}$$

and recalling the deduction that lead to (5), we know that $\frac{\hat{g}\hat{A}}{\hat{B}-1}=-1$. Substituting this into (7) gives

$$\frac{\hat{f}_n}{\hat{f}} = \hat{g}\hat{B}^n-(\hat{B}^n-1)$$

and substituting $\hat{g}$, $\hat{A}$, and $\hat{B}$,

$$= \left(\frac{1}{2r+1}\frac{\sin(r\omega+\omega/2)}{\sin(\omega/2)}\right)\cos(2r\omega+\omega)^n-\cos(2r\omega+\omega)^n+1.$$

This may be simplified while maintaining an upper bound.

$$\begin{aligned}\left|\frac{\hat{f}_n}{\hat{f}}\right| &= \left|\left(\frac{1}{2r+1}\frac{\sin(r\omega+\omega/2)}{\sin(\omega/2)}\right)\cos(2r\omega+\omega)^n-\cos(2r\omega+\omega)^n+1\right| \\ &\le \left|\left(\frac{1}{2r+1}\frac{\sin(r\omega+\omega/2)}{\sin(\omega/2)}\right)\cos(2r\omega+\omega)^n\right|+|\cos(2r\omega+\omega)^n|+1 \\ &\le \left|\left(\frac{1}{2r+1}\frac{\sin(r\omega+\omega/2)}{\sin(\omega/2)}\right)\cos(2r\omega+\omega)^n\right|+2 \\ &\le \left|\frac{1}{2r+1}\frac{\sin(r\omega+\omega/2)}{\sin(\omega/2)}\right|+2\end{aligned} \tag{8}$$

Next we prove by induction that

$$\left|\frac{1}{2r+1}\frac{\sin(r\omega+\omega/2)}{\sin(\omega/2)}\right|+2 \;\leq\; 3$$

or equivalently

$$|\sin(r\omega+\omega/2)| \;\leq\; (2r+1)|\sin(\omega/2)|. \tag{9}$$

Starting from (9), the base case for induction is $r=0$:

$$\begin{aligned} |\sin(0\cdot\omega+\omega/2)| &\leq (2\cdot 0+1)|\sin(\omega/2)| \\ |\sin(\omega/2)| &\leq |\sin(\omega/2)| \end{aligned}$$

which is clearly true. The inductive step (letting $r=k+1$):

$$\begin{aligned} & |\sin((k+1)\,\omega+\omega/2)| \\ = \;& |\sin(k\omega+\omega+\omega/2)| \\ = \;& |\sin(k\omega+\omega)\cos(\omega/2)+\cos(k\omega+\omega)\sin(\omega/2)| && \text{(by angle addition)} \\ \leq \;& |\sin(k\omega+\omega)||\cos(\omega/2)|+|\cos(k\omega+\omega)||\sin(\omega/2)| \\ \leq \;& |\sin(k\omega+\omega)|+|\sin(\omega/2)| && (\text{since } 1\leq\cos(u)\leq 1) \\ \leq \;& |\sin(2(k+1)\,\omega/2)|+|\sin(\omega/2)| \\ \leq \;& 2(k+1)|\sin(\omega/2)|+|\sin(\omega/2)| && (\text{since } |\sin(u)|\leq u) \\ \leq \;& (2(k+1)+1)|\sin(\omega/2)| \end{aligned}$$

which establishes

$$\begin{aligned} |\sin(r\omega+\omega/2)| &\leq (2r+1)|\sin(\omega/2)|. \\ \therefore \quad \left|\frac{1}{2r+1}\frac{\sin(r\omega+\omega/2)}{\sin(\omega/2)}\right|+2 &\leq 3 \end{aligned}$$

and thus, from (8),

$$\begin{aligned} \left|\frac{\hat{f}_n}{\hat{f}}\right| &\leq 3 \\ \frac{|\hat{f}_n|}{|\hat{f}|} &\leq 3 \\ |\hat{f}_n| &\leq 3|\hat{f}| \\ |\hat{f}_n\, e^{i\omega t}| &\leq 3|\hat{f}|. \end{aligned}$$

We have now established the function $3|\hat{f}|$ as an upper bound for $|\hat{f}_n\, e^{i\omega t}|$ which is integrable over $\omega\in[-\pi,\pi]$ (since $\hat{f}$ is integrable). The conditions are now met to apply the dominated convergence theorem and exchange the integral and the limit.

$$\begin{aligned} \frac{1}{2\pi}\int_{-\pi}^{\pi}\lim_{n\to\infty}(\hat{f}_n\, e^{i\omega t})\,d\omega &= f && \text{(from (6))} \\ \lim_{n\to\infty}\left(\frac{1}{2\pi}\int_{-\pi}^{\pi}\hat{f}_n\, e^{i\omega t}\,d\omega\right) &= f && \text{(dominated convergence theorem)} \\ \lim_{n\to\infty}(\mathcal{F}^{-1}(\hat{f}_n)) &= f && \text{(definition of inverse DTFT)} \\ \lim_{n\to\infty}(f_n) &= f \end{aligned}$$

This proves Theorem 1. □