A fast, simple, and parallelizable deconvolution algorithm for real-time applications

Daniel Williams*

ABSTRACT

We present a fast, simple, and parallelizable deconvolution algorithm for the real-time deblurring of one- or two-dimensional signals (i.e. images) degraded by defocus or bokeh-like blur. The proposed algorithm runs in linear-time and performs significantly faster than other popular deconvolution methods tested, bringing the deblurring time down to under 10ms for full-HD images. It has a simple software implementation, requiring no Fourier transforms or dynamic memory allocation. Its parallel design makes it especially suitable for GPU acceleration. For one-dimensional noise-free signals, the algorithm is proven to converge exactly to the original un-blurred signal.

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

1. INTRODUCTION

Image deconvolution (or deblurring) is often considered a time-consuming computation. In this paper, we focus on optimizing 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 particularly common case of this degradation is that of an out-of-focus blur (sometimes called *bokeh*). In this case, the point spread function, g, is disk-like (Figure 1). It is on this case that we focus our attention. Our goal is to compute f given only g, defined by a single parameter r, the radius of the disk. We will assume that the added noise, s, is small.

Many methods have been proposed to solve such a problem, such as 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.[4,5] In contrast, Wiener deconvolution is a fast but complex fast-fourier-transform (FFT) based approach that operates in the frequency domain.[4,5] Ideally, we would want a fast algorithm which is also simple and parallelizable. This is the basis of our method (which we'll call FAST-METHOD), albeit, at the sacrifice of generality.

Although we have only mentioned image deblurring, these algorithms (including ours) may 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.

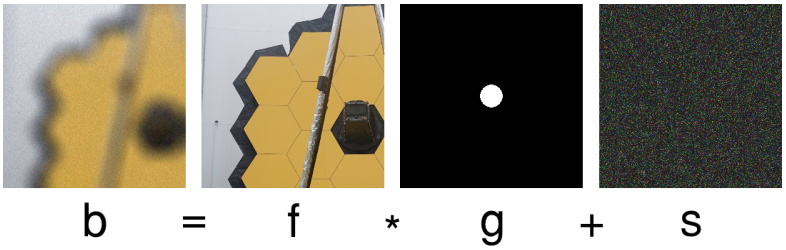

Figure 1. Basic model of defocus-blurred image. Observed degraded image (b) is created by convolving the ideal image (f) with a disk-like point spread function (g) and adding noise (s).

*Email: danieljwilliams1114@yahoo.com

2. THE ALGORITHM FOR 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 increasingly accurate approximations $f_n[i]$. The FAST-METHOD works in five steps:

1. Add 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, $f_0[i]$. These steps are illustrated in Figure 2. The algorithm may also be written as a single equation:

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

For each iteration, the new value at each index may be computed independently of each other index, allowing for massive parallelization. Additionally, assuming the absence of noise, the use of a box blur kernel, and that we know r, this method converges exactly to $f[i]$ in the limit of n. A proof of this is given in Appendix A.

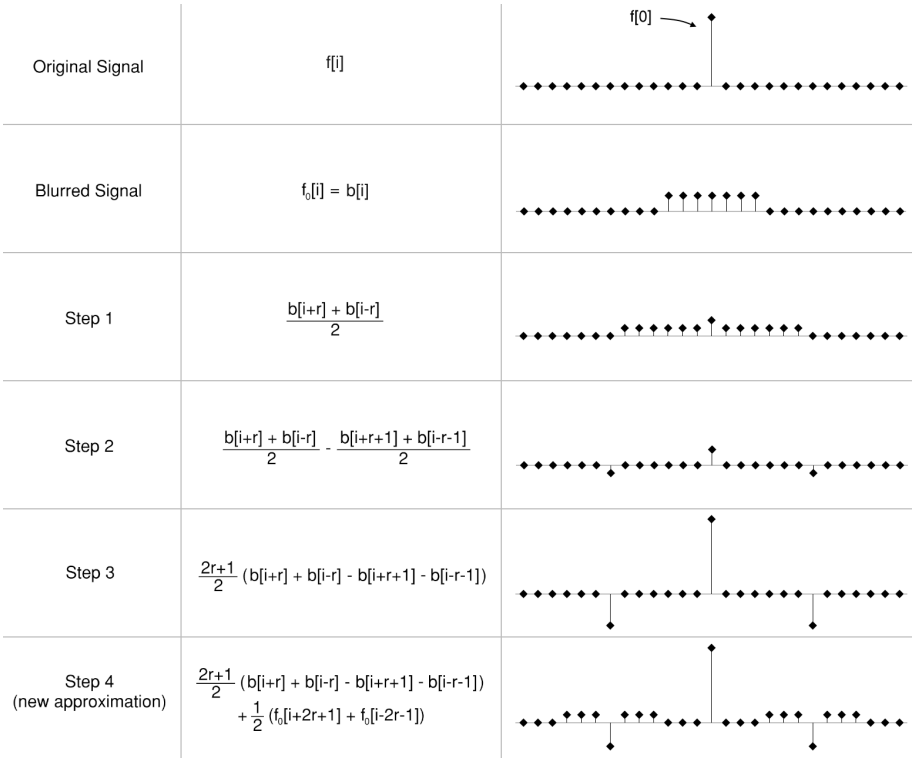

Figure 2. Illustration of steps of the first iteration of FAST-METHOD applied to an example Dirac delta in one dimension. The points represent discrete samples of the signal. The sharp central peak of the original signal is recovered.

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 more clever implementation would clamp the array indexes to within the bounds of the signal, effectively extending its borders. The `for` loop may also be parallelized over `i`.

3. THE ALGORITHM FOR 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. Add 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 experimentally determined. 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.

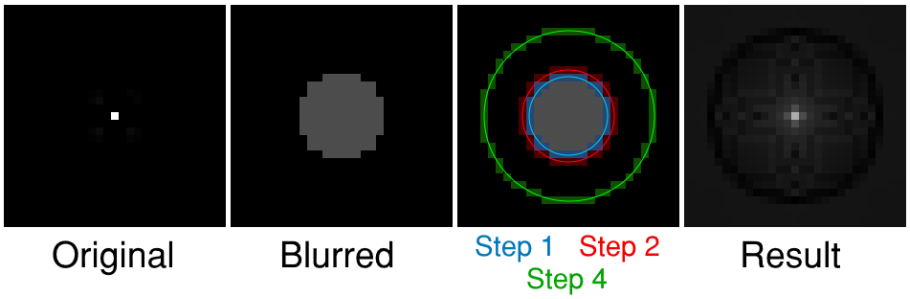

Figure 3. Graphic illustration of the steps of the first iteration of Fast-Method applied to a blurred point. The images are exaggerated for clarity. The result mostly recovers the original image, with some additional ringing effects.

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)
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 proper 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 $O(n\,r)$ where n is the number of pixels in the image, and r is the blur radius. 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 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 the first iteration alone typically produces useful results.

4. RESULTS

We'll compare our FAST-METHOD against two other representative algorithms:

- Richardson-Lucy, chosen for its accuracy, parallelizability, and iterative spatial-domain approach.[1,2,4,5]
- Wiener deconvolution, chosen for its speed and FFT-based approach.[3,4,5]

Both algorithms are popular in many applications.

As our measure of error, we use the normalized root mean square error (NRMSE) defined on an ideal signal a and its reconstructed 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 multichannel color images, each channel is normalized first (by subtracting the average), then added in the summation. 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 In one dimension

Figure 4 shows the convergence of FAST-METHOD when applied to flourish.wav†, a musical audio file with about 17,600,000 samples. The audio was blurred by radius $r = 1000$ using the following point spread function:

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

It was then saved to a new file and deconvolved using FAST-METHOD. The blurred audio file sounded muffled upon playback. After 21 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. Even after 650 iterations, it continued to converge to the original undegraded audio.

†Copyright Microsoft. flourish.wav was converted from the MIDI file, flourish.mid, which is distributed with many versions of Windows OS.

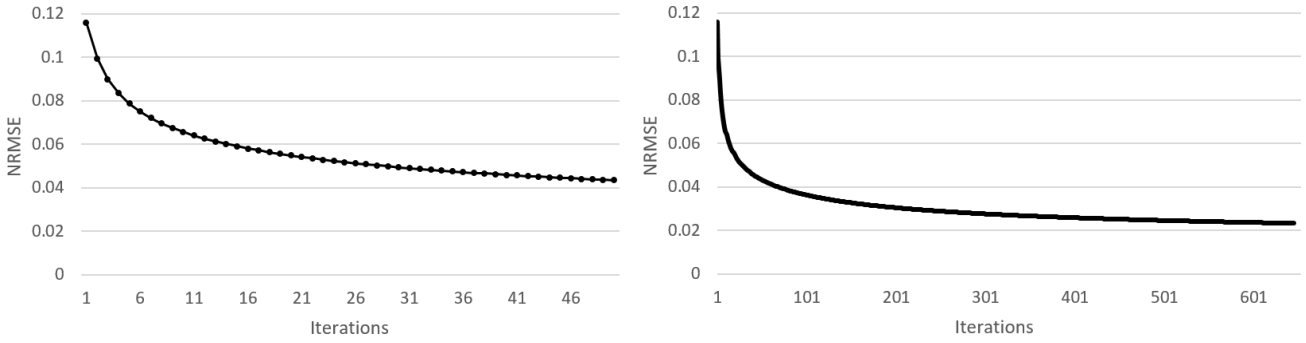

Figure 4. NRMSE vs iterations for Fast-Method applied to audio file flourish.wav. The graph on the left shows the first 50 iterations of the graph on the right.

Figure 5 shows the effect of Fast-Method being applied to a blurred step function at various iterations. On each iteration, the jagged artifacts introduced by the algorithm are reduced in amplitude and spread farther from the origin. After 101 iterations, the step function appears to have been completely recovered, confirming that Fast-Method has very accurate convergence.

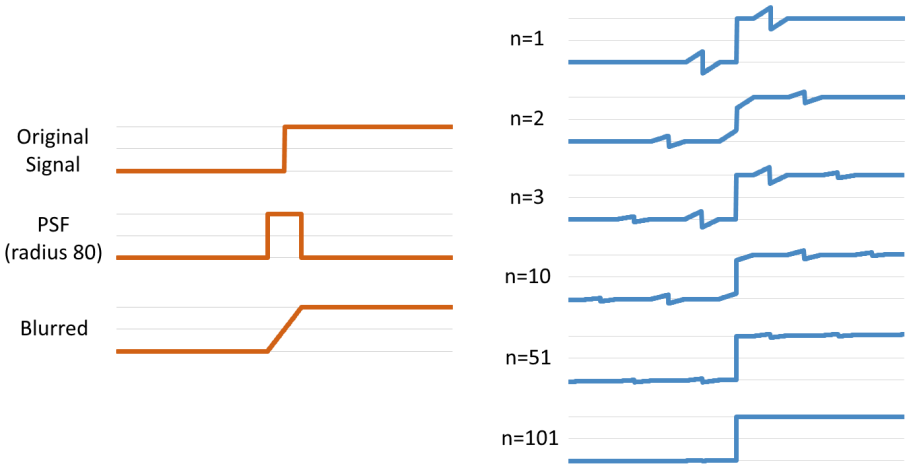

Figure 5. Fast-Method applied to a step signal at various iterations.

4.2 In two dimensions

For thorough comparison, each of the three algorithms, Richardson-Lucy, Wiener, and Fast-Method, have been implemented in multiple languages and parallelized. For Richardson-Lucy, we use a parallelized Java implementation, a parallelized OpenCL implementation for GPU acceleration, and Mathematica's‡ built-in `ImageDeconvolve`[4] function. For Wiener deconvolution, we use a parallelized C++/OpenCV implementation[3], a parallelized OpenCV/CUDA implementation for GPU acceleration, and Mathematica's `ImageDeconvolve` function. For the 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 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 and SNR for Wiener deconvolution, were tuned to produce the lowest

‡Wolfram Mathematica 13.1 using function `ImageDeconvolve` with `Method→"RichardsonLucy"` and `Method→{"Wiener",SNR}` for Richardson-Lucy and Wiener deconvolution respectively.

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 the algorithms produced improvement over the unprocessed blurred image. For the FAST-METHOD, a single iteration ($n = 1$) typically produced the best result.

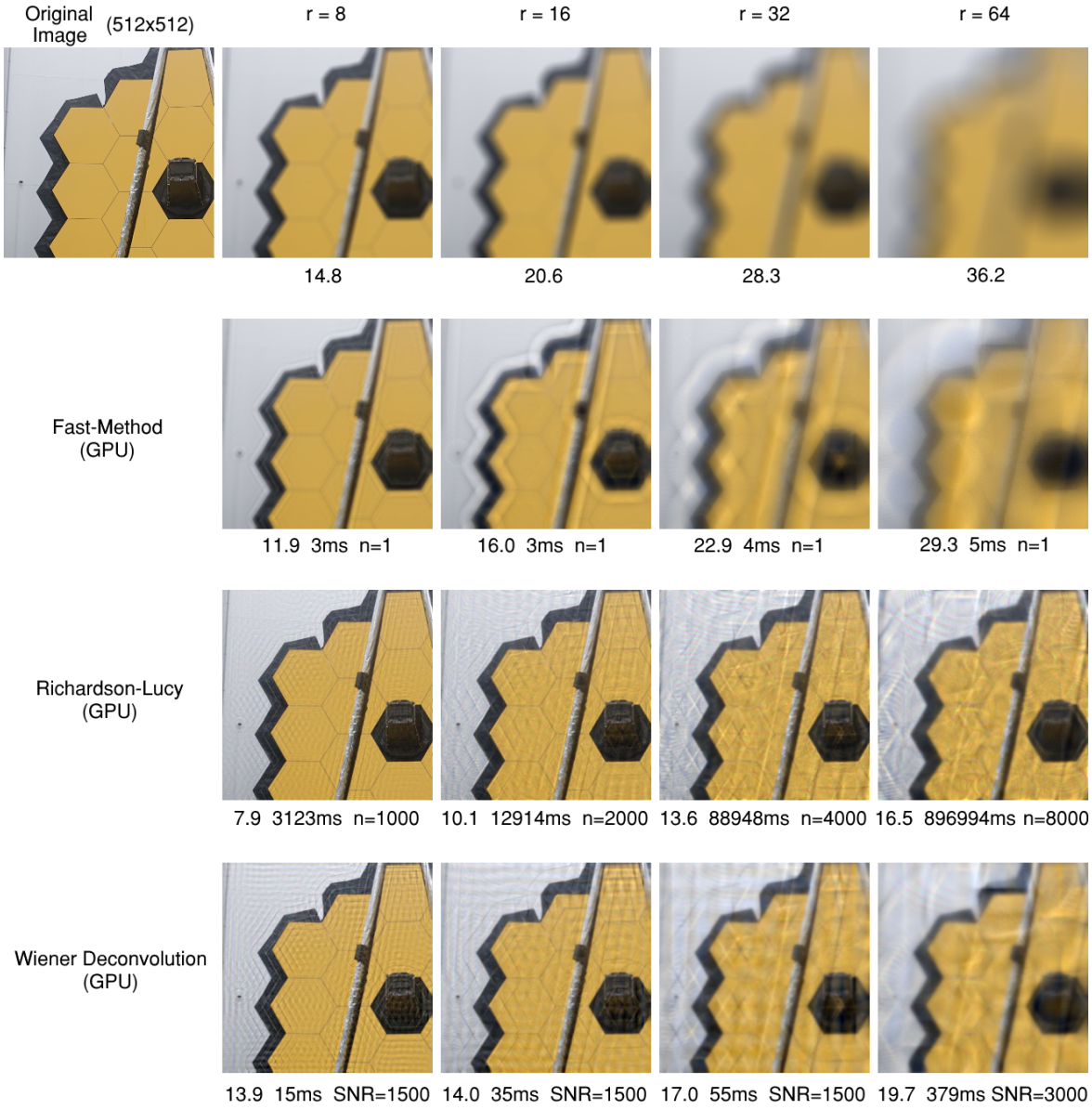

Figure 6. Comparison of deblur quality with execution time for each method at various blur radiuses. The top row is the blurred image prior to processing, captioned by their NRMSE. Under each result is NRMSE, time, and parameters for the algorithm applied.

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.

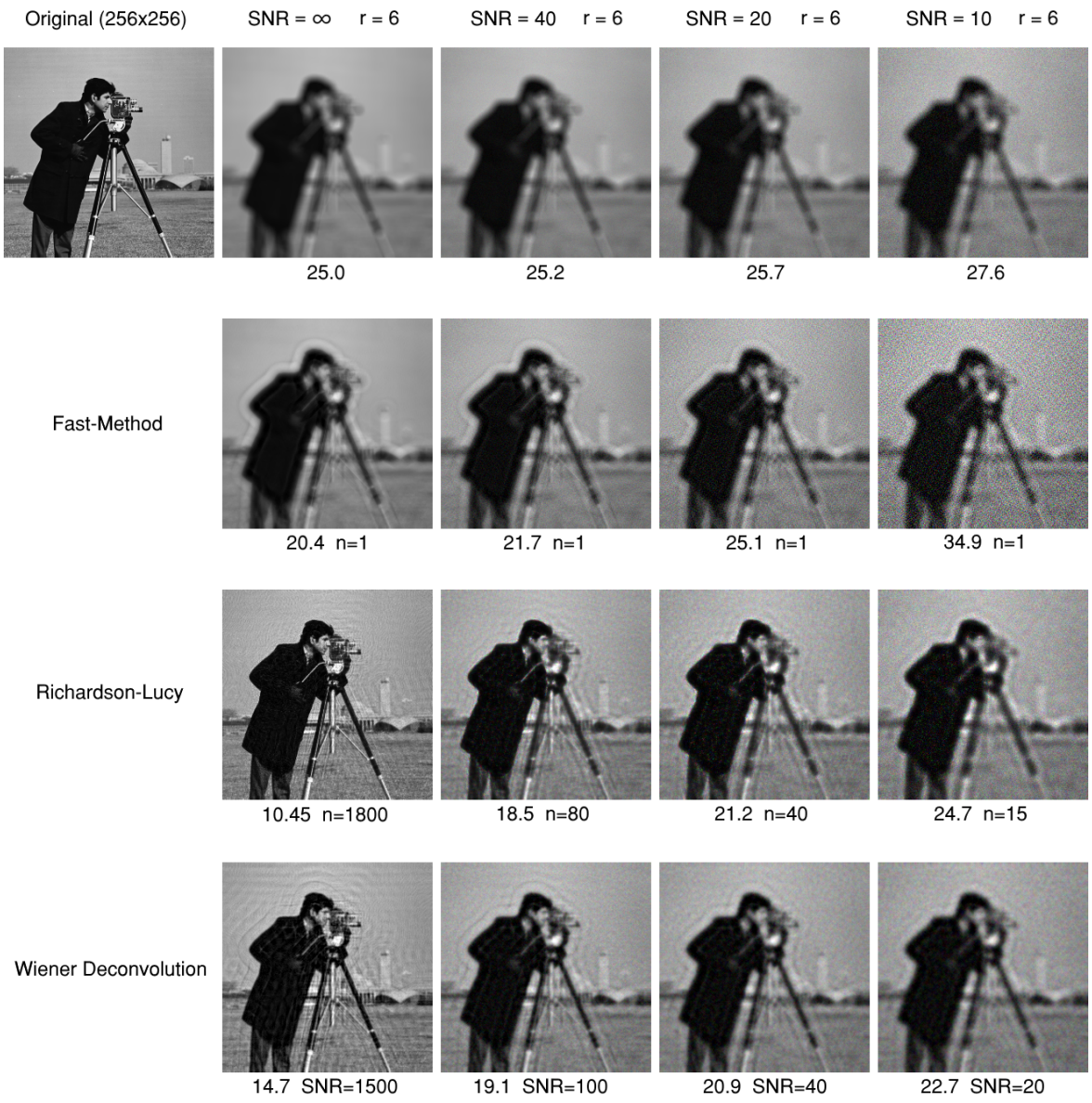

Figure 7. Comparison of deblurring methods in the presence of additive gaussian noise. The top row is the image prior to processing captioned with NRMSE. Under each result is NRMSE and the parameters used for the algorithm. Image courtesy of MIT.

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 three 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 JPG with resolution 5472×3080. The radius of the blur was then estimated by brute-force—deblurring with various radiuses until the best setting was found. All the algorithms restored the text to a legible degree. FAST-METHOD outperformed the other methods by at least an order of magnitude, while Wiener deconvolution produced the best result visually. In this case, the accuracy of Richardson-Lucy was limited by its lengthy runtime (about 25 minutes).

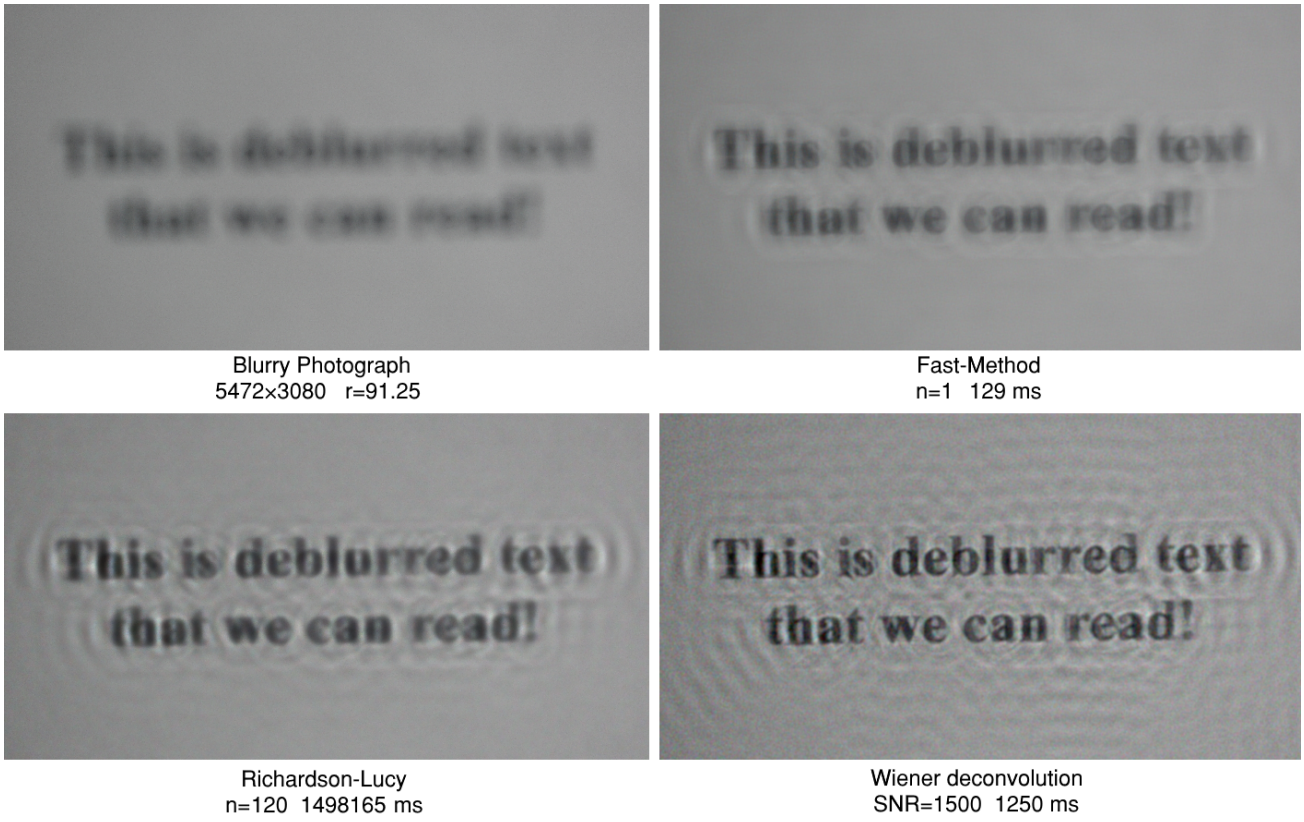

Figure 8. Comparison of deblurring methods on an out-of-focus photograph.

Tables 1 and 2 show the performance of each implementation for each algorithm on an image blurred by $r = 16$ and $r = 32$, respectively. The timing metrics were gathered by running each algorithm on a 1920×1080 color image with each algorithm's parameters adjusted to produce the same quality result (i.e. nearly equivalent NRMSE). The OpenGL implementation of Fast-Method was by far the fastest implementation. Presumably, this is because the design of Fast-Method fits aptly into the OpenGL shader pipeline, where each pixel's color is computed independently of all others, and that processed pixels need not be copied back to main memory between iterations. An OpenGL implementation of Richardson-Lucy and Wiener deconvolution was not obtained due to their complexity.

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)
Java (CPU)	39416 ms	-	156 ms
OpenCL (GPU)	1685 ms	-	10 ms
OpenGL (GPU)	-	-	3 ms
C++/OpenCV (CPU)	-	115 ms	-
C++/OpenCV (GPU)	-	84 ms	-
Mathematica	15350 ms	711 ms	-

Table 2. Same as Table 1, but the test image was blurred by $r = 32$.

Implementation	Richardson-Lucy (n=31)	Wiener deconvolution (SNR=1500)	Fast-Method (n=1)
Java (CPU)	1212118 ms	-	2762 ms
OpenCL (GPU)	32982 ms	-	69 ms
OpenGL (GPU)	-	-	19 ms
C++/OpenCV (CPU)	-	815 ms	-
C++/OpenCV (GPU)	-	594 ms	-
Mathematica	123930 ms	5116 ms	-

Finally, we analyze the performance of Fast-Method in the application of real-time video processing. Table 3 shows the performance of Fast-Method when deblurring 1080p color video. To achieve real-time speeds, the OpenGL implementation was used. The video was stored as a compressed 15,000kbps MP4 file. Each frame was read off a solid-state drive, decoded, deblurred, and displayed on the screen in sequence. Pairs of frames were processed in parallel to allow for simultaneous decoding and deblurring. Our implementation was able to process video at up to 118 frames per second—much faster than real-time. Even as the blur radius was increased, the additional execution time of Fast-Method did not exceed the time required to read and decode each frame. In other words, the video processing rate was not limited by the execution of Fast-Method.

Table 3. 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.

Blur radius	Frame rate	Total frame time	Fast-Method time
r=9	118 FPS	8.5 ms	2.9 ms
r=21	117 FPS	8.5 ms	2.9 ms
r=40	116 FPS	8.6 ms	6.0 ms

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 Tables 1 and 2, 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 may be performed in real-time.

(ii) *Real-time audio processing:* Deblurring imagery is 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 video conferencing or recovering attenuated sound.

(iii) *Starting approximation for another algorithm:* Many iterative deconvolution algorithms, including Richardson-Lucy, require a starting approximation. Often, the blurred image is simply used as a starting approximation. The more accurate the starting approximation, the quicker good results are produced. Our Fast-Method provides a good first approximation to jump-start lengthier iterative methods.

(iv) *Estimation of blur radius:* The general problem of estimating a point spread function is called *blind deconvolution*.[5] If we are restricted to only disk-like blurs, then the Fast-Method offers a convenient way to estimate the radius of such a disk blur. Rather than applying a mathematical technique which extracts the blur radius, we can simply try all radiuses by brute force—visually inspecting each. The algorithm is sufficiently quick that a user may interactively adjust the blur radius—viewing the results in real time—until the image appears in-focus. Indeed, this method was used to determine the blur radius of many images in the Results.

6. CONCLUSIONS

We have proposed a new deconvolution algorithm which significantly outperforms other methods tested. Thanks to the simplicity and parallelizability of the algorithm, it was relatively easy to implement in a variety of languages including Java, OpenCL, and OpenGL. Using the OpenGL implementation, we were able to deblur 1080p color video at up to 118 frames per second. Although it performs quickly, it produced less accurate results than other popular deconvolution methods tested. However, in one dimension, we mathematically prove and experimentally verify that our method converges exactly to the un-blurred solution in the absence of noise.

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 sharp-edged point spread functions, such as ellipses or squares.

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 to handle. By applying the 1D Fast-Method to a 2D image in the direction of the blur, we would expect a decent result, like that achieved in one dimension in the Results.

Although we have given a proof of the convergence of the Fast-Method in one dimension, we have failed to find a proof in two dimensions. And indeed, it seems that iteratively applying the algorithm in two dimensions gives negligible improvement. This lack of iterative improvement in two dimensions seems out of place, and may be worth further investigation.

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] Wolfram Research, "ImageDeconvolve", Wolfram Language & Documentation Center, 2012, https://reference.wolfram.com/language/ref/ImageDeconvolve.html (27 July 2023).

[5] Fliegel, K., Janout, P., Bednář, J., Krasula, L., Vítek, S., et al., "Performance evaluation of image deconvolution techniques in spacevariant astronomical imaging systems with nonlinearities," Proc. SPIE 9599, 959927 (2015)

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. The rest of this proof will apply only to the one-dimension 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. □