Use Of Sharp Image Content to Enhance Sharpness of Other Image Areas

Henry Dietz and Hunter Durkee
Department of Electrical and Computer Engineering, University of Kentucky; Lexington, Kentucky

Abstract

Many lenses have significantly poorer sharpness in the corners of the image than they have at the center due to optical defects such as coma, astigmatism, and field curvature. In some circumstances, such a blur is not problematic. It could even be beneficial by helping to isolate the subject from the background. However, if there exists similar content in the scene that is not blurry, as happens commonly in landscapes or other scenes that have large textured regions, this type of defect can be extremely undesirable. The current work suggests that, in problematic circumstances where there exists visually similar sharp content, it should be possible to use that sharp content to synthesize detail to enhance the defectively blurry areas by overpainting. The new process is conceptually very similar to inpainting, but is overpainting in the same sense that the term is used in art restoration: it is attempting to enhance the underlying image by creating new content that is congruous with details seen in similar, uncorrupted, portions of the image. The `kongsub` *(Kentucky's cONGruity SUBstitution) software tool was created to explore this new approach. The algorithms used and various examples are presented, leading to a preliminary evaluation of the merits of this approach. The most obvious limitation is that this approach does not sharpen blurry regions for which there is no similar sharp content in the image.*

Introduction

The current work is one of a multitude of approaches to computational enhancement of image detail[1]. The goal here is not to increase image size, as is done by various upscaling or super-resolution algorithms, but to increase the level of detail visible. Algorithms with this effect are commonly referred to as methods for image restoration, sharpening, or deblurring. The field of deblurring algorithms is particularly crowded, with a multitude of methods published[2][3][4]. Blur in an image often is the result of either motion of the subject relative to the camera or poor focus, and those causes are the target of most test datasets and algorithms[5]. However, those are not the only real-world causes of blur.

The type of blur which is the primary target for improvement here is caused by optical defects such as coma, astigmatism, and field curvature. These defects are often suffered by low-cost or ultra-wide lenses and are distinctive in that the impact on blur varies dramatically with distance off the optical axis; it is common that image content that is roughly centered is entirely unaffected. The key attribute is the fact that similar scene content is imaged with very different degrees of blur for different parts of the image. This both makes the blur more visually disruptive and implies that other portions of the scene have textures which could be suitably applied to repair the blurred area.

Figure 1. *Sample images shot with Fujian 35mm f/1.7*

This type of distinctive blur characteristic is evident in the pair of images shown in Figure 1, both captured using a Fujian 35mm $f/1.7$. This lens probably was not designed to cover an APS-C sensor. It not only suffers significant vignetting, but also suffers from strong field curvature. The top image is a capture of a flat, sharp, photo. This 2D subject makes the field curvature of the Fujian lens quite obvious. Also noteworthy is the fact that focus was not set on the center, but about halfway to the edge; this results in a fairly apparent ring of sharpness. The lower photo was captured directly of an outdoor scene. It also shows a ring of best focus, although somewhat less blatantly due to the 3D scene. Perhaps most interesting is that both these images began

as 24MP captures using a Sony NEX-7, yet the image defects being discussed are clearly visible here despite the images having been scaled from 6000×4000 to just 600×400 pixels.

The earliest deblurring approaches generally applied simple filters to increase local contrast, such as unsharp masking or perhaps deconvolution. Most more recent methods have centered on using trained AI, so-called "deep deblurring" methods[5], to replace portions of the content with credible sharp details. For example, using AI that can recognize faces, a person's slightly blurry eye in an image might be replaced with a sharp image of an eye including crisply-rendered eyelashes. The image detail is thus greatly increased, but the result is not necessarily very similar to the details of the scene that was photographed. The implicit guarantee is that the details synthesized look like things in the training set images, but the image being enhanced was not in that set.

The motivation for the current work is to guarantee that the details synthesized look like things in the image being processed. There are a variety of inpainting algorithms that share the goal of synthesizing content that is derived from and consistent with the unaltered portions of the image being enhanced[6][7]. However, inpainting replaces regions with content that can be arbitrarily different from the original content. For example, the image content generated typically does not have the same average color or brightness as the content it replaced. To be precise, that was never the goal of traditional inpainting: inpainting synthesizes textures to credibly connect the areas around a damaged or removed region of the image. In contrast to traditional inpainting, our goal here might be best described as *overpainting*: generating enhanced textures that refine the image content rather than completely replacing it. This concept has been used before in `kremy` (KentuckY's Raw Error Modeler)[8][9] to reduce noise and provide a modest improvement in resolution. The main difference between `kremy` and the approach here is that `kremy`'s improvements are constrained by the concept of the captured image being corrupted only by noise, whereas here overpainting is far more aggressive, based on overpainting higher-resolution detail over blurry areas that have lower resolution.

The current work is distinguished by two main features:

- The type of blur targeted for improvement is one often ignored in other work: blur due to use of poorly-corrected optics. Typically, this defect is most severe in the corners of a capture due to optical flaws including coma, astigmatism, and field curvature. However, it also can happen, for example, when a smudge on the front element of a lens smears the corresponding portion of the image. How undesirable such blur is depends very strongly on the scene content. These defects become particularly unpleasant in things like landscapes, images where a 2D object occupies a significant fraction of the frame, or other scenes that have large textured regions because the texture of the region changes (smears) as the viewer's attention is moved across the frame. These are the types of images emphasized in evaluation of the new approach.
- The overpainting texture synthesis is not constrained by a noise model, but is instead driven by a model of how much blur is present. Precisely measuring blur is less straightforward than one might expect[10], but even crude approximations can be used to distinguish which regions should be sources of texture and which should be overpainted.

Figure 2. *Grayscale versions of sample images*

The Prototype Implementation

The prototype discussed in this paper is a crude first implementation of this new approach. It is not capable of producing results of the highest quality that this general approach should allow, nor is it using a particularly efficient set of algorithms for processing. The intent is that the current work will establish that the concepts underlying this approach are sound and future work will dramatically improve the implementation.

Kentucky's cONGruity SUBstitution tool, `kongsub`, is a quite simple C++ program using the OpenCV (OPEN Computer Vision) library[11]. It is not tuned for performance, but does use OpenMP[12] directives to obtain parallel execution of some operations on multi-core processors. Runtime for processing a small image, such as the 600×400 examples shown, is on the order of 10 minutes using a typical multi-core desktop computer. That speed is acceptable for a first prototype, but clearly must be improved upon for the methods used by `kongsub` to become widely adopted. The following subsections detail how the images are processed by `kongsub` and expose which tasks slow the processing.

Figure 3. *Tonally inverted Laplacians of the sample images*

Figure 4. *Blur maps for the sample images*

Convert the Image to Grayscale

After reading the input color image, the first processing step is the production of a grayscale version. There are several reasons why the color information is removed, including to facilitate the next processing step. However, the main reason is simply that the goal is to synthesize textures with which the blurry image content can be overpainted – often, the same texture will appear multiple times with a scene with slightly different coloration. Thus, the conversion to grayscale can be considered as a mechanism to give priority to textural matches over coloration matches. This transformation is trivially done using OpenCV's `cvtColor()` function to produce the images seen in Figure 2.

Compute a Blur Map

The first versions of `kongsub` took two images as input: a color image to be improved and a monochrome image to serve as a blur map. The map is intended to encode blur by some version of the principle that the greater the blur around the corresponding pixel in the image to be improved, the brighter the pixel value in the blur map. Ideally, the pixel values in the blur map would directly correspond to a scaled blur diameter (or CoC – circle of confusion). There are many different algorithms that follow that principle, from simple Laplacian edge detection to frequency-domain analysis to identify the dominant spatial frequency of the image content at each pixel. Having the blur map as an input allowed manual experimentation with various ways to construct the map, which quickly revealed that:

- Algorithmic construction of a map perfectly identifying blur due to lens defects, as opposed to deliberate defocus, motion blur, or low-spatial-frequency scene content, is an open problem when given just a single image. It is relatively easy to characterize a lens in this way by examining multiple images, especially using controlled scenes (e.g., test charts), but the impact of optical defects can change significantly with commonly-changed shooting parameters – such as focus distance.
- Even a very imprecise blur map can be useful.

The current version of `kongsub` merely accepts an image from which it creates a blur map using a very straightforward procedure – which could be dramatically improved upon.

The first step in computing the blur map is edge detection. The current software does this by computing a Laplacian to create an image giving the divergence of the gradient of the monochromatic image's tones. The Laplacian is largest where the edge is strongest, but the strongest edge implies the smallest CoC. Thus, it is most natural to think of the tonally-inverted Laplacian, as shown in Figure 3, as being darker where the CoC is smaller.

In theory, the Laplacian could be directly used as a map. However, the Laplacian features are essentially well-defined edges. If a well-defined edge exists in a particular region within an image, then by definition that region is not an area degraded

by blur. In other words, detection of a sharp edge implies the lack of a sharp edge immediately adjacent is also a reliable sampling of the scene. Thus, the edges detected in the original image represent the cores of local regions not needing sharpening, and simply thickening the detected edges can provide a better blur map. A simple way to thicken edges and reasonably combine them where edges meet is to apply a blur operation to the Laplacian. In `kongsub`, this is done using a series of applications of OpenCV's `GaussianBlur()` operation to convolve with Gaussian functions of various sizes. The result of that is then histogram equalized using OpenCV's `equalizeHist()` operation. Figure 4 shows the blur maps resulting from this thickening of the identified edges.

Are the blur maps shown in Figure 4 representing the right attribute? It is easy to see that the blur maps have indeed identified where each image is sharp, and the doughnut-shaped ring of sharpness for the top image is clear. The bottom image does not have as obvious a doughnut shape because the scene it photographed was far from flat. Sharpness varies wildly in this scene as the interplay of field curvature and the 3D depths to scene elements vary throughout the image. There is also a sharper edge in the bottom image due to the high contrast between the blackness of the fence and the light color of the grass – that edge arguably does not belong in the map. Ideally, the blur map should have each pixel shaded in proportion to the size of the CoC at that position; the blurred Laplacian does not ensure that, but can be produced using a relatively simple computation.

Find Textural Patch Matches

Ideally, the blur map would tell us precisely how blurry each pixel is, and that information could be used to to to find sharper content that matches when blurred by the same amount. For example, suppose position X_b, Y_b has a blur diameter of D; then a perfectly sharp (blur diameter 1) patch at X_s, Y_s should be blurred by a factor of D before attempting a textural match with the patch at X_b, Y_b. Unfortunately, the method currently used to compute the blur map does not provide that information, so instead and arbitrary amount of blur is applied to the original image to make an image to search for matches.

Although OpenMP parallelism is used, nearly all of the computational effort is expended in patch matching. For each pixel location, the current version of `kongsub` searches for the patch match which is the best match with the highest sharpness. A viable alternative would be to use the weighted combination of good patch matches, but preliminary experiments did not find that beneficial. For each position X_b, Y_b, each pixel value from the unblurred version of the best-matching patch at X_m, Y_m is summed into a result image and the weight for each pixel is also summed in a weight image by which the final result is divided. Given a 3×3 patch size, each final pixel is thus the weighted average of 9 texture matches and the original value. The result of this is a "raw" overpainted image, as shown in Figure 5.

This pattern matching and synthesis step could be dramatically sped-up by using pattern hashing, which also could improve the pattern match quality by allowing larger patterns and/or more blur levels to be considered. The problem of matching similar textures within an image is closely related to the problem of finding matching features from which alignments of multiple images can be computed. For example, the concepts of the scale-invariant feature transform (SIFT)[13] and alternative techniques[14] could be applied to this problem. The main difference would be that in feature matching many candidate point patterns are discarded to increase selectivity of the match, whereas here relatively poor (ambiguous) matches can still contribute to the recovery of a sharper texture. However, investigation of such approaches is beyond the scope of this introductory exploration.

Figure 5. *Raw overpainted versions of sample images*

Similarly, there is the potential to use trained AI methods to generate the pattern structures to overpaint. Instead of training on a set of images which may be unrelated to the image to be enhanced, it may be feasible to train a neural network exclusively using the sharp regions of the image to be enhanced. Investigation of such methods is beyond the scope of the current work.

Normalize the Overpainted Image

While the overpainted image should be very similar to the original, the weighted averaging process does not incorporate any mechanism that would ensure colors are maintained. As a result, the "raw" overpainted image will often have significantly different, often more muted, coloration than the original. This is repaired by converting the RGB image into CIE L*a*b* colorspace and combining the Lightness value from the overpainting with the a* and b* values from the original image. Figure 6 shows the result of this color correction.

Figure 6. *CIE L*a*b* color correction of overpainting*

Figure 7. *Final enhanced versions of sample images*

As a final filter, the Laplacian is computed for the overpainted image and the original and overpainted images are merged using the Laplacian values as weighting factors. The sharper the detected edges, the heavier the weighting given in the weighted summation of original and color-corrected overpainted images. Weighted transparency of the overpainting has the effect of partially restoring local contrast; without this step, the overpainted image will often have excessively dark regions. The final result is shown in Figure 7.

While the very preliminary results presented here show relatively modest levels of sharpening, the images are significantly sharper in the blurriest regions while still appearing organic to the scene. Of course, blurry regions for which no similar scene content was sharply recorded will not be enhanced as credibly, although this was not a major issue in the test cases processed. The obvious exception is the handling of the sky in the upper left corner of the top image and the upper right corner of the bottom image. This comes from the fact that rather than being evenly lit, the vignetting of the lens imposes a sharp enough gradient in the corners to be incorrectly treated as blurred edges. Correcting the vignetting before this processing appears to remedy this issue. In summary, there is good reason to believe that a more sophisticated implementation of this general approach could be highly effective.

Conclusion

Inpainting replaces undesired, missing, or damaged portions of an image with synthetic content that is consistent with its surroundings. In contrast, overpainting is the act of applying paint over another layer of paint or surface, which is fundamentally different in that overpainted content need not completely replace the underlying image content. In earlier work[9], it has been shown that inpainting-like texture synthesis can be effective as an overpainting technique to reduce noise and as a side effect mildly enhance sharpness. Here texture-based overpainting is directly used to increase sharpness driven by a (crude) empirically-computed map of blur in the original image.

The motivating problem is a class of blur which is empirically common, but often omitted from study when exploring sharpening or deblurring algorithms: lens defects which cause uneven sharpness across the image, most often with blur increasing farther off the optical axis. This class of blur has the interesting property that there is often similar scene content imaged with varying levels of blur, thus making it likely that imposing textures from sharper areas onto blurry ones can produce a good approximation to a sharp rendering of the actual scene content.

Viability of this approach was confirmed by creating and testing a prototype open source C++ implementation, `kongsub` (Kentucky's cONGruity SUBstitution), which is available from `https://aggregate.org/DIT/KONGSUB`. Significant future work is need to make a useful production tool.

References

[1] Yunliang Qi, Yang Zhen, Sun Wenhao, Lou Meng, Lian Jing, Zhao Wenwei, Deng Xiangyu, and Ma Yide, "A comprehensive overview of image enhancement techniques," Archives of Computational Methods in Engineering, pp. 1-25, 2021

[2] J. Biemond, R. L. Lagendijk and R. M. Mersereau, "Iterative methods for image deblurring," in Proceedings of the IEEE, vol. 78, no. 5, pp. 856-883, May 1990, doi: 10.1109/5.53403

[3] Ruxin Wang and Dacheng Tao, "Recent Progress in Image Deblurring," arXiv preprint arXiv:1409.6838, 2014

[4] A. Mahalakshmi and B. Shanthini, "A survey on image deblurring," 2016 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2016, pp. 1-5, doi: 10.1109/ICCCI.2016.7479956

[5] K. Zhang, T. Wang, W. Luo, W. Ren, B. Stenger, W. Liu, H. Li and M. H. Yang, "MC-Blur: A comprehensive benchmark for image deblurring," IEEE Transactions on Circuits and Systems for Video Technology, Volume 34, Number 5, pp. 3755-3767, 2023

[6] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles and Coloma Ballester, "Image inpainting," SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 417 - 424, doi: 10.1145/344779.344972, 2000

[7] Christine Guillemot and Olivier Le Meur, "Image Inpainting: Overview and Recent Advances," EEE Signal Processing Magazine, Volume 31, Issue 1, January 2014, pp. 127 - 144, doi: 10.1109/MSP.2013.2273004

[8] Henry Gordon Dietz, "Refining raw pixel values using a value error model to drive texture synthesis" in Proc. IS&T Int'l. Symp. on Electronic Imaging: Image Processing: Algorithms and Systems XV, 2017, pp 56 - 66, doi: 10.2352/ISSN.2470-1173.2017.13.IPAS-084

[9] Henry Dietz, "An improved raw image enhancement algorithm using a statistical model for pixel value error" in Proc. IS&T Int'l. Symp. on Electronic Imaging: Computational Imaging, 2022, pp 151-1 - 151-6, doi: 10.2352/EI.2022.34.14.COIMG-151

[10] D. Shaked and I. Tastl, "Sharpness measure: towards automatic image enhancement," IEEE International Conference on Image Processing 2005, Genova, Italy, 2005, pp. I-937, doi: 10.1109/ICIP.2005.1529906.

[11] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000

[12] OpenMP Architecture Review Board, "OpenMP Application Programming Interface," Version 6.0, November 2024, `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-6-0.pdf`

[13] David G. Lowe, "Object recognition from local scale-invariant features," Proceedings of the International Conference on Computer Vision, Volume 2, pp. 1150-1157, doi: 10.1109/ICCV.1999.790410, 1999

[14] L. Juan and O. Gwun, "A comparison of sift, pca-sift and surf," International Journal of Image Processing (IJIP), 3(4), pp.143-152, 2009