A Comparison of Various Edge-Aware Filtering Techniques

Gilbert Fan

December 2021

Github Link: https://github.com/hopestrikestwice/A-Comparison-of-Various-Edge-Aware-Filtering-Techniques

1 Abstract

I have implemented four edge-aware filtering techniques (Bilateral Filter, Guided Filter, Domain Transform, and Local Laplacian Filter) in python in order to analyze their advantages and disadvantages. Overall, the local Laplacian filter produces the highest quality images at the cost of time. The domain transform also creates sharp images after denoising, but contains staircase artifacts if severely scrutinized. The guided filter performs better than an unaugmented bilateral filter and seems best for a general use case.

2 Introduction

Given the large amount of different edge-aware filtering techniques in existence, it is hard to immediately grasp their differences to decide when to use each one. I have selected four techniques (Bilateral Filter, Guided Filter, Domain Transform, and Local Laplacian Filter) and reimplemented them in order to analyze how their algorithmic differences reflect in resulting image quality.

3 Methods

I implemented the following methods in python, with the goal of testing their edge-aware filtering capabilities and differences.

3.1 Bilateral Filter

The bilateral filter is a basic method for smoothing images while preserving edges, or areas where pixel values change dramatically. It applies a filter pixel-by-pixel, weighing the neighboring pixels by spatial distance and value similarity to the center pixel. The idea is represented by the following equation from [1]:

$$bf(I)_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in N(\mathbf{p})} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in N(\mathbf{p})} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)$$

Here, I is the image. p, q are positions in the image, and can be represented as [row, column] coordinates. G_{σ_s} is a Gaussian performed on spatial distance, and G_{σ_r} is a Gaussian performed on the difference between values of the image at position p and q.

The general smoothing is given by the spatial distance, which is similar to a Gaussian filter, but the addition of a weight based on value difference helps this method become edge-aware. A pixel in a flat, similar area will have a neighborhood with similar weight contributions from G_{σ_r}, making G_{σ_s} the main determinant of weight difference and reducing the filter to basically a Gaussian filter. On a sharp edge pixel, the neighbors on the wrong side of the edge will get weighted significantly lower by G_{σ_r}, lending more influence to the similar value on the other side of the edge.

For my implementation, I used the Piecewise Bilateral algorithm from [2], which is a faster implementation of the bilateral filtering algorithm by turning it into two convolutions for every discretized pixel intensity, and then using these to interpolate the actual value.

3.2 Guided Filter

The guided filter produces an output based on a linear transformation of a guide image, using coefficients computed from the input image. The equation representing this algorithm is as follows:

$$q_i = a_k I_i + b_k, \forall i \in w_k$$

In contrast to the bilateral filter, which computes output values pixel-by-pixel, the guided filter computes results patch by patch. Here, the patch is a window w_k of user-defined size. Each pixel position is respresented by i, and the pixel's value in the guide image is I_i. a_k and b_k are coefficients for this window determined by the input image. Over the whole image, multiple windows will overlap to help produce the output q_i value for a single pixel, and this is determined by averaging the a_k, b_k value over the windows:

$$q_i = \bar{a_i} I_i + \bar{b_i}$$

The coefficient a is found by dividing the covariance between guide image I and input image p by the sum of the variance of intensity values of I plus a constant ϵ, which the user provides. b is found by subtracting a times the average values of the guide image I from the average values of the input image

p. Additionally, while the guided filter allows for two image inputs, I mostly performed the filter with the same image as both the guide and input, as besides joint bilateral filtering, the other algorithms only take one image as input, and my goal was to see the differences between algorithms under the same conditions. This makes a become variance of I divided by the sum of the variance of I and ϵ.

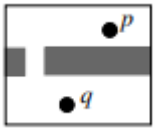
The intuition behind this method's edge-aware ability is made clear through the coefficients. When there is large variance of I, such as on an edge or highly detailed area, ϵ becomes relatively small, so a approaches 1 and b approaches 0. This causes a large weight on keeping I, so relatively little blurring is performed. On the other hand, flat areas such as a wall or large shadow, which may include noise, have relatively little variance of I, so a approaches 0 and b approaches the mean of the input image over the window, allowing us to smooth the area more strongly.

3.3 Domain Transform

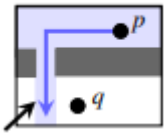
The domain transform attempts to speed up edge-aware filtering by performing filtering in lower dimensions. Given a 2D RGB image, which can be interpreted as a 2D signal or in 5D space, we would like to transform it into just 2D space, allowing us to apply a fast, 2D filter to the image at once. Our goal is to find an isometric transformation, which keeps the distances in the original space the same as the transformed space. Recall the bilateral filter uses spatial and intensity distance for its edge-aware smoothing ability, which is reflected in the image's 5D representation. Thus, preserving distance is essential for maintaining edge-awareness. Unfortunately, no generally applicable transformation from 5D to 2D exists.

However, [3] has shown that an isometric mapping from 4D to 1D (or any c+1 dimension to 1 dimension) does exist, and that it preserves the geodesic distance between pixels. If a 1D signal was represented as a curve, the geodesic distance is the distance between points on the curve. The process in [3] essentially unfolds the curve in 1D space, preserving arc length distance. Thus, given a 1D RGB signal, we can transform it into 1D space and then apply a 1D filter, which will be much faster than applying the filter in its original 4D space.

There is still one last problem in this method, as the ability to transform a 1D signal does not immediately generalize to 2D signals. Instead, we will take our 1D filters and apply them to the image horizontally, and then vertically, in succession. However, one round of these two passes does not immediately give pixels the information they would have if we had applied a filter to the original 2D image. Because our filters are edge-aware, by definition they do not transfer information over strong edges. Consider the following two pixels, p, q from a figure in [3]:

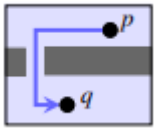


The grey line is a strong edge. After one round of using our 1D filter horizontally and vertically, the p pixel would be "seen" by all the blue areas:



However, if the original image was filtered normally, the neighborhood of p would have included the pixel q and the area behind the strong edge.

This is solved by running an additional iteration of horizontal and vertical passes, resulting in the following:

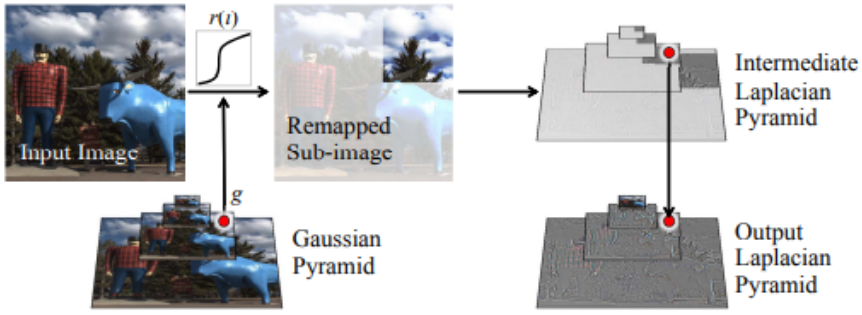


Thus, we see the more two-pass iterations of 1D filtering we do, the more accurately our algorithm models a 2D filter. Of course, running more iterations also takes more time. [3] suggests three iterations is generally best, and so this is the number I used to produce my results.

3.4 Local Laplacian Filter

The local Laplacian filter is essentially reconstructing the image from a Laplacian pyramid. A Laplacian pyramid is typically constructed by taking the difference between levels in a Gaussian pyramid, upsampling smaller levels so sizes are compatible. A Gaussian pyramid is built by downsampling images with a Gaussian kernel, which is spatially invariant as it does not change no matter what pixel location it is working on, it just applies the same to the entire image. Thus, the Gaussian filter is not edge aware at all.

Yet the Gaussian pyramid is still used as a core component of the local Laplacian filter. Instead of building a Laplacian pyramid of the entire image directly from the Gaussian pyramid, we use the Gaussian pyramid to guide in remapping windows of the original image around the corresponding pixel to the Gaussian pyramid, and create local Laplacian pyramids from just that area. Consider the following picture from [5]:



After creating the Gaussian pyramid from the original input, we go pixel-by-pixel within the Gaussian pyramid. At each pixel g, we use its value in the pyramid to remap a sub-image window of the original image by comparing the distance between the value of g and the value of each pixel in the sub-image to a user-defined value σ. If this distance is less than σ, the pixel is remapped as a detail, and otherwise it is remapped as an edge.

The remapped sub-image is then used to create an intermediate Laplacian pyramid, and the corresponding pixel g in this intermediate pyramid is copied to the final, output, Laplacian pyramid. Thus, we construct the output Laplacian pyramid pixel-by-pixel, until finally, we can collapse it to get the output image.

4 Results

My resulting images are in Github under the results folder. For brevity, I will only show notable results under the Analysis section.

5 Analysis

I will be using the following original pictures for comparison (note all pictures are compressed for the pdf):

Lamp 

Treepile 

The lamp image is from the homework. I use it as my main comparison as I believe it does the best to illustrate my points. I took a selection of my own images I ran the algorithms on to confirm my observations, and their results are in the results folder.

5.1 Bilateral Filter

I used a bilateral filter as a baseline for a generic edge-aware filtering, so I will not go into much of its analysis here. Instead, the analysis of other algorithms will draw back to the bilateral filter results.

Here is a close-up of the bilateral lamp image:

5.2 Guided Filter

In the paper, the guided filter claims to be better at preserving structures than a bilateral or joint bilateral, but I have not been able to notice this to a significant degree. This may be because the images I have ran the filtering on were all taken in dark lighting and do not provide areas with enough contrast. In [4], the example they used to demonstrate this point was a feathering task performed on a black-white image, so my images have comparatively low contrast.

The guided filter does seem to have better retention of small details and thin lines, as the straw ends in the middle and bottom of the lamp are much better preserved. This is likely because when the bilateral detects and "edge" in this area, the kernel may be large enough to detect other straws at the same time, and the bilateral will assign them higher weights due to their similar intensities, blurring the original pixel somewhat. However, when the guided filter runs across this area, it detects the many detailed straws as high variance, and will increase the weight of its a coefficient, increasing the weight of the guide image I, which gets used directly in the output, instead of getting slightly blurred by the similar pixels across multiple edges nearby.

Another big difference between the guided and bilateral filtering algorithms is that the guided filter just takes an average when it performs its blurring, and works over a box blur, where the kernel weighs every pixel the same. Bilateral filtering has a different weight for every pixel, and the weights of the kernel changes as it moves through the image. In some of the flatter, noisy areas, such as the shadows in the lamp image, the two filters seem to produce slightly different results.

Here is a close up of the guided filter:

The noise-reducing ability of the bilateral and guided filter seem similar. The flatter region is much more smooth in both cases. Compared to the bilateral close-up, however, the protruding strands of the guided filter image seem to have less of a halo, though one still exists.

5.3 Domain Transform

The domain filter is considered the worst at preserving structure by [4]. This is noticeable when zooming into the results. I think this is best illustrated by the Treepile image:

The straight edge of a paper crane, which was retained in other algorithms, is now riddled with dents and ridges. This is likely because of the domain filter's nature of needing to do a horizontal-vertical pass in 1D, instead of a single 2D filter across the image, and so these spots appear in a kind of staircase-like pattern along edges.

While the image does not look nearly as good as the other algorithms when zoomed in, the full result appears to maintain its details more sharply than guided or bilateral filtering. The lamp seems to pop out, and the result looks crisp, so long as it is not zoomed in on.

At a slightly less magnified level, the edges of a domain filter seem to carry almost no halo effect at all, despite the jagged nature of its edges:

5.4 Local Laplacian Filter

For the Laplacian filter, I had to perform the filter over downsampled images, as even their optimized sequential algorithm took many hours to run for a single, downsampled image. The paper says the speed can be improved by parallelizing their algorithm, but I did not have the time to implement this.

Even with the downsampling, the laplacian filter seems to be the best at preserving details and thin edges. The straws within the middle and bottom of the lamp look better than the other algorithms'. However, while it is fine at denoising the image, at higher parameter levels an effect appears in the background:

As you can see, parts that do not have a strong edge contrast with the background, such as the plates, quickly begin to merge with the background. This is noticeable to a lesser degree in the first Laplacian filter image, and it is clear the the upper left plate is much more blended than in other algorithms' results.

Here is a close-up of the laplacian result:

From this, we can see there is no halo effect whatsoever on the image's edges. This is likely due to the algorithm constructing laplacian pyramids on a point-by-point basis, so the points across the edge can better separate themselves from the edge, compared to the bilateral filtering still blurring them into the edge due to spatial proximity.

6 Conclusion

Overall, I think the local laplacian filter produced the best results over all the images I tested, as it had no halo and the details remained the clearest. Its only downside of blurring weak edges is not prominent enough at the low strength needed for denoising images. A more important downside, however, is its speed. My implementation only begins to a take a reasonable amount of time on images downsampled to less than a few hundred pixels on each side. On a full image, the algorithm easily takes more than several hours.

If computation time is important, then the domain transform appears to have an acceptable speed with very good results, only taking about 30 minutes for the pictures I took. Of course, the downside here is that there are jagged

artifacts along the edges if the results need to be scrutinized extremely closely.

If this is unacceptable, or an even faster time is desired, then the guided filter recommended. The guided and bilateral filter appear to have a good amount of overlap in their features, but the guided filter performs just noticeably better in almost all aspects compared to a generic bilateral filtering algorithm.

7 Citations

[1] CHEN, J., PARIS, S., AND DURAND, F. 2007. Real-time edge-aware image processing with the bilateral grid. ACM Transactions on Graphics (Proc. SIGGRAPH) 26, 3.

[2] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'02, pages 257–266, New York, NY, USA, 2002. ACM.

[3] Eduardo S. L. Gastal and Manuel M. Oliveira. "Domain Transform for Edge-Aware Image and Video Processing". ACM Transactions on Graphics. Volume 30 (2011), Number 4, Proceedings of SIGGRAPH 2011, Article 69.

[4] He, K., Sun, J., amp; Tang, X. (2013). Guided Image Filtering. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(6), 1397–1409. https://doi.org/10.1109/tpami.2012.213

[5] Paris, S., Hasinoff, S. W., amp; Kautz, J. (2015). Local laplacian filters. Communications of the ACM, 58(3), 81–91. https://doi.org/10.1145/2723694