Fast Trilateral Filtering

Tobi Vaudrey and Reinhard Klette

The *.enpeda..* Project, The University of Auckland, Auckland, New Zealand

Abstract. This paper compares the original implementation of the trilateral filter with two proposed speed improvements. One is using simple look-up-tables, and leads to exactly the same results as the original filter. The other technique is using a novel way of truncating the look-up-table to a user specified required accuracy. Here, results differ from those of the original filter, but to a very minor extent. The paper shows that measured speed improvements of this second technique are in the order of several magnitudes, compared to the original or LUT trilateral filter.

1 Introduction

Many smoothing filters have been introduced, varying from the simple mean and median filtering to more complex filters such as anisotropic filtering [2]. These filters aim at smoothing the image to remove some form of noise.

The trilateral filter [1] was introduced as a means to reduce impulse noise in images. The principles of the filter were based on the bilateral filter [7], which is an edge-preserving Gaussian filter. The trilateral filter was extended to be a gradient-preserving filter, including the local image gradient (signal plane) into the filtering process. Figure 1 demonstrates this process using a geometric sketch. This filter has the added benefit that it requires only one user-set parameter (the starting bilateral filter size), and the rest are self-tuning to the image.

The original paper [1] demonstrated that this filter could be used for 2D images, to reduce contrast of images and make them clearer to a user. It went on to highlight that the filter could be used to denoise 3D images quite accurately. Recent applications of trilateral filtering have shown that it also very applicable

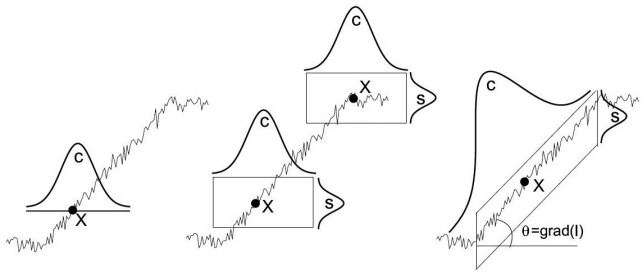

Fig. 1. Illustration of the filtering process using (from left to right) unilateral (Gaussian), bilateral, or trilateral filtering (figure from [1]).

to biomedical imaging [9]. It decreases noise while still preserving fine details. The trilateral filter has also been used to create residual images (illumination invariant images), to increase the quality of optical flow and stereo matching [8]. Using only one pass produces sufficient results.

Unfortunately, the filter is very slow and requires large local search regions when the image has a low gradient. This issue only gets worse with increasing image sizes, as the filter is not linear in running time. This makes large 2D or 3D images very slow to compute. Also, for smaller images, the use of this filter for real-time applications (such as driver assistance systems and security cameras) is limited.

This paper presents a novel numerical approximation for the trilateral filter. The proposed approach increases the speed of the filter dramatically (e.g., from hours down to seconds), while still maintaining high accuracy. The filter does not use parallel processing, but can still be parallalised to further increase speed. The approach requires one additional user parameter, required accuracy. Furthermore, we have implemented the trilateral algorithm for standard 2D images, which has been made publicly available [6].

We first introduce the original trilateral filter, followed by a simple speed up technique that does not generate data loss (look up tables). We then present our novel approach, using kernel truncation based on required data accuracy. This is followed by results demonstrating the speed improvements, and the differences in results to the original filter.

2 Definition of Trilateral Filter

An image is defined by $f(\mathbf{x}) \in \mathbb{R}^n$ (n = dimensionality), where $\mathbf{x} \in \Omega$ is the pixel position in image domain Ω. Generally speaking, an n-D (n-dimensional) pixel-discrete image has an image domain defined as, $\emptyset \subset \Omega \subseteq X_n \subset \mathbb{N}^n$ (X_n is our maximum discrete index set of the image domain in dimension n). A smoothing operator will reduce an image to a smoothed version of itself, specifically $S(f) = s$, where s is in the same image domain as f. To introduce the trilateral filter, we must first define the bilateral case; we will then go on to define the traditional trilateral filter using this notation.

2.1 Bilateral Filter

A bilateral filter is actually an edge-preserving Gaussian filter. Of course, the same technique could be used with any type of simple filter (e.g., median or mean). Offset vectors $\mathbf{a}$ and position-dependent real weights $w_1(\mathbf{a})$ (spatial smoothing) define a local convolution, and the weights $w_1(\mathbf{a})$ are further scaled by a second weight function w_2 (colour/magnitude smoothing), defined on the differences $f(\mathbf{x}+\mathbf{a}) - f(\mathbf{x})$:

$$s(\mathbf{x}) = \frac{1}{k(\mathbf{x})} \int_\Omega f(\mathbf{x}+\mathbf{a}) \cdot w_1(\mathbf{a}) \cdot w_2\left[f(\mathbf{x}+\mathbf{a}) - f(\mathbf{x})\right] \mathrm{d}\mathbf{a} \tag{1}$$

$$k(\mathbf{x}) = \int_\Omega w_1(\mathbf{a}) \cdot w_2\left[f(\mathbf{x}+\mathbf{a}) - f(\mathbf{x})\right] \mathrm{d}\mathbf{a}$$

Function $k(\mathbf{x})$ is used for normalization. The weights w_1 and w_2 are defined by Gaussian functions with standard deviations σ_1 (colour) and σ_2 (spatial), respectively (another filter can be substituted, but will provide different results). The smoothed function s equals $S_{BL}(f)$. The bilateral filter requires a specification of parameters σ_1, σ_2, and the size of the used filter kernel $2\mathbf{m}+1$ in f ($\mathbf{m}$ is the *half kernel size* and is n-dimensional). Of course, the size of the kernel can be selected using σ_1 and σ_2.

2.2 Trilateral Filter

The trilateral filter is a "gradient-preserving" filter. It aims at applying a bilateral filter on the current plane of the image signal. The trilateral case only requires the specification of one parameter σ_1. At first, a bilateral filter is applied on the derivatives of f (i.e., the gradients):

$$g_f(\mathbf{x}) = \frac{1}{k_\nabla(\mathbf{x})} \int_\Omega \nabla f(\mathbf{x}+\mathbf{a}) \cdot w_1(\mathbf{a}) \cdot w_2\left(||\nabla f(\mathbf{x}+\mathbf{a}) - \nabla f(\mathbf{x})||\right) \mathrm{d}\mathbf{a} \quad (2)$$

$$k_\nabla(\mathbf{x}) = \int_\Omega w_1(\mathbf{a}) \cdot w_2\left(||\nabla f(\mathbf{x}+\mathbf{a}) - \nabla f(\mathbf{x})||\right) \mathrm{d}\mathbf{a}$$

To approximate $\nabla f(\mathbf{x})$, forward differences are used, and more advanced techniques (e.g., Sobel gradients, 5-point stencil) are left for future studies. For the subsequent second bilateral filter, [1] suggested the use of the smoothed gradient $g_f(\mathbf{x})$ [instead of $\nabla f(\mathbf{x})$] for estimating an approximating plane

$$p_f(\mathbf{x}, \mathbf{a}) = f(\mathbf{x}) + g_f(\mathbf{x}) \cdot \mathbf{a} \quad (3)$$

Let $f_\triangle(\mathbf{x}, \mathbf{a}) = f(\mathbf{x}+\mathbf{a}) - p_f(\mathbf{x}, \mathbf{a})$. Furthermore, a neighbourhood function

$$N(\mathbf{x}, \mathbf{a}) = \begin{cases} 1 & \text{if } |g_f(\mathbf{x}+\mathbf{a}) - g_f(\mathbf{x})| < c \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

is used for the second weighting. Parameter c specifies the adaptive region and is discussed further below. Finally,

$$s(\mathbf{x}) = f(\mathbf{x}) + \frac{1}{k_\triangle(\mathbf{x})} \int_\Omega f_\triangle(\mathbf{x}, \mathbf{a}) \cdot w_1(\mathbf{a}) \cdot w_2(f_\triangle(\mathbf{x}, \mathbf{a})) \cdot N(\mathbf{x}, \mathbf{a}) \, \mathrm{d}\mathbf{a} \quad (5)$$

$$k_\triangle(\mathbf{x}) = \int_\Omega w_1(\mathbf{a}) \cdot w_2(f_\triangle(\mathbf{x}, \mathbf{a})) \cdot N(\mathbf{x}, \mathbf{a}) \, \mathrm{d}\mathbf{a}$$

The smoothed function s equals $S_{TL}(f)$.

Again, w_1 and w_2 are assumed to be Gaussian functions, with standard deviations σ_1 and σ_2, respectively. The method requires specification of parameter σ_1 only, which is at first used to be the diameter of circular neighbourhoods at $\mathbf{x}$ in f; let $\overline{g}_f(\mathbf{x})$ be the mean gradient of f in such a neighbourhood. The parameter for w_2 is defined as follows:

$$\sigma_2 = \beta \cdot \left| \max_{\mathbf{x} \in \Omega} \overline{g}_f(\mathbf{x}) - \min_{\mathbf{x} \in \Omega} \overline{g}_f(\mathbf{x} \right| \quad (6)$$

($\beta = 0.15$ was recommended in [1]). Finally, $c = \sigma_2$.

3 Numerical Speed Improvements

In the previous section, we defined the trilateral filter in a continuous domain. But as we are all aware, the numerical approximation needs to be implemented in real-life. And obviously, this filter takes a lot of processing to work. This section aims at showing how numerical implementations are improved dramatically.

In practice w_1 (the spatial weight) from Equation (2) can be pre-calculated using a look-up-table (LUT), as it only depends on σ_1, $\mathbf{m}$ (kernel size), and $\mathbf{a}$ (offset from central pixel). As this function is Gaussian, the LUT is computed as follows:

$$W_1(\mathbf{i}) = \exp\left(\frac{-2\left\|\mathbf{i}\right\|^2}{\sigma_1^2}\right) \tag{7}$$

where $\mathbf{0} \leq \mathbf{i} \leq \mathbf{m}$ (usually $\mathbf{i} \in \mathbb{N}^n$, but can also approximate vectors in $\mathbb{Q}^n$ using interpolation). W_1 is used by simple referencing using $W_1(|\mathbf{a}|)$,[1] which approximates a quarter of the Gaussian kernel (as the Gaussian function is symmetric). Unfortunately, the intensity weight w_2 can not use a look up table, as it depends on the local properties of the image.

Similar principles can be applied to Equation (3). In this equation, w_1 depends on a local adaptive neighbourhood A, depending on the magnitude of the gradients. However, the function is only dependent on the distance from the central pixel $\mathbf{a}$, and since the maximum of A is known, the LUT can be computed as in Equation (7), but where $\mathbf{0} \leq \mathbf{i} \leq \max(A)$. Again, w_2 depends on local information, no LUT can be used. This approach is called the *LUT-trilateral filter*.

From here, to improve speed, there need to be numerical approximations. The presented approach is a smart truncation of the kernel to a defined accuracy $\varepsilon \in \mathbb{Q}^+$, and $0 < \varepsilon < 1$. We know that the function is Gaussian so we shall use this for our truncation. If we want to ignore any values below ε, then only values above this should be used:

$$\varepsilon \leq \exp\left(\frac{-2\left\|\mathbf{i}\right\|^2}{\sigma_1^2}\right) \quad \text{which leads to} \quad \left\|\mathbf{i}\right\| \leq \sigma_1\sqrt{-\frac{1}{2}\ln(\varepsilon)} = T$$

where T is the threshold. (Note that $\ln(\varepsilon)$ is strictly negative, so T is strictly positive.) In practice, this means that we can compute a look up table as defined in Equation (7), where $0 \leq ||\mathbf{i}|| \leq T$. This approach could be applied to a bilateral filter, but does not really benefit it. However, when dealing with the trilateral filter, this reduces the number of equations dramatically (as the largest kernel size is equal to the size of the smallest dimension $\min_n(X_n)$ in the image). This truncation will increase the error, but only by, at most, $\varepsilon\left(\min_n(X_n)\right)^2$. We call this method the *fast-trilateral filter*. Note that this filter has only two parameters (which are both logical); σ_1 (the initial kernel diameter) and ε (the required accuracy).

[1] $|\mathbf{a}|$ is here short for $(|a_1|, \ldots, |a_n|)$, and $||\mathbf{a}||$ is the L_2-norm.

Note that this does not exploit any parallel processing, but is open to massive parallel processing potential, as every pixel is independent within the iteration of trilateral filtering. This is especially noticeable for GPU programming, where the truncated LUT can be saved to texture memory [5].

4 Experimental Results of Filter

We have implemented the trilateral algorithm for standard 2D images, which has been made publicly available [6]. The experiments of this section were performed on a Intel Core 2 Due 3G Hz processor, with 4GB memory, on a Windows Vista platform. Parallel processing was not exploited (e.g., OpenMP or GPU). Of course, further speed improvements can be gained by doing so.

4.1 Dataset

We illustrate our arguments with the 2005 and 2006 Middlebury stereo datasets [4], provided by [3]. We selected a sample set to use for our experiments: *Art, Books, Dolls, Reindeer* and *Baby1*. For each image from this dataset, we use the full resolution image (approx. 1350×1110). We then scale down the image by 50% in both directions, and repeat this 5 times (i.e., 50%, 25%, 13%, 6%, and 3% of original image size), see right part of Figure 2 for example of images used. This allows us to demonstrate running times for differing image sizes.

4.2 Comparison of Running Time

Figure 2 shows the running times of the algorithms on the *Art* images. The results compare two σ_1 values of 3 and 9, and the fast trilateral filter uses $\varepsilon = 10^{-12}$. There is obviously a massive improvement when using trilateral-LUT compared to the original (especially with larger images). With smaller images, the improvement is under 1 magnitude, but increases quickly up to around 1 magnitude improvement (see 1390×1110 results). There is no reason to use the original method instead of the LUT, as there is no accuracy loss with the LUT (the memory usage is negligible compared to calculating image pyramids).

The fast-trilateral filter shows a massive improvement over the other methods (except 43×34, which is not a practical image size). The improvement only gets better as the size of the image increases; for the largest image size the difference is 46 hours (original) and 5 hours (LUT), compared to 86 seconds for the fast-trilateral filter. That is a dramatic decrease (several orders of magnitude) in computation time.

From these results, we can infer that the improvements will only get better when extending the filter to 3-dimensions (e.g., filtering noisy 3D-meshes), as the number of pixels (or voxels) increases further.

When using the fast-trilateral filter, the user selects the required accuracy. The less accuracy wanted, the faster the filter runs. The comparison in Figure 2 is for the highest accuracy ($\varepsilon = 10^{-12}$), which highlights the improvement over the other filters. To show the effect of reducing the accuracy, compared to running

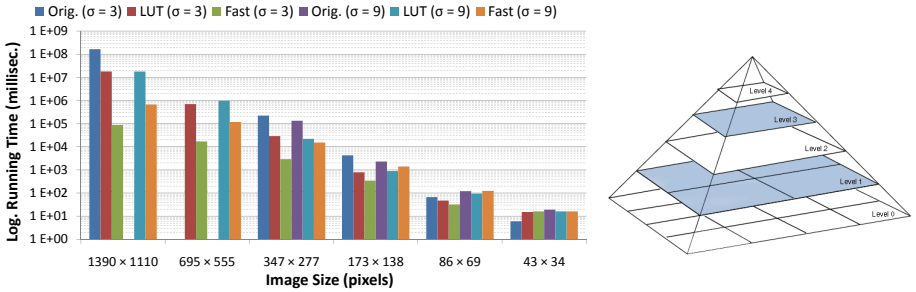

Fig. 2. Running times for *Art* image (left) for different scales of an image (right); displayed in $\log_{10}$ scale. Note: original was not run for $\sigma = 9$ on largest image, nor on second largest image (due to time). For the fast trilateral results, $\varepsilon = 10^{-12}$.

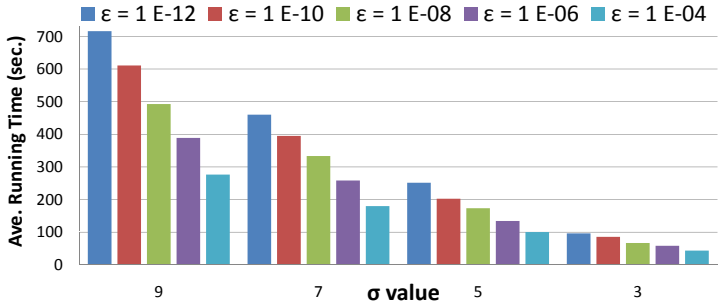

Fig. 3. Average running times for fast-trilateral filter on the dataset of images at maximum resolution. Shows the results for varying kernel sizes σ_1 and accuracy ε.

time, we ran the filter across the dataset (at maximum resolution, i.e., approx. 1350×1110) and averaged the running times. Figure 3 shows the results of the fast-trilateral filter for this test using varying kernel sizes (σ_1). This graph shows that the improvements with decreasing accuracy are linear (within each σ_1). A point to note is that when using $\sigma_1 = 9$, the difference in running time goes from 700 seconds ($\varepsilon = 10^{-12}$), down to 290 ($\varepsilon = 10^{-4}$). The next section demonstrates that the results from the fast-trilateral filter are very close to the original (and LUT) filter, showing that this speed improvement is for almost no penalty.

4.3 Accuracy Results

A difference image d is the absolute difference between two images,

$$D(s, s^*) = d \quad \text{with} \quad d(\mathbf{x}) = |\, s(\mathbf{x}) - s^*(\mathbf{x}) \,| \tag{8}$$

where s is the fast-trilateral result, and s^* is the result from the LUT-trilateral (original) filter. Using this we can calculate the maximum difference $\max_{\mathbf{x} \in \Omega}(d)$, in the image. An example of difference images can be seen in Figure 4. This figure illustrates that there are some subtle differences between the LUT-trilateral filter

and the fast-trilateral filter, but they are actually minor. When using $\varepsilon = 10^{-12}$, the differences are too negligible to even count. As for using $\varepsilon = 10^{-4}$, the differences are still very small (maximum error is still less than half an intensity value).

To assess the quality of an image, there needs to be an error metric. A common metric is the *Root Mean Squared* (RMS) *Error*, defined by

$$E_{RMS}(d) = \sqrt{\frac{1}{|\Omega|} \sum_{\mathbf{x} \in \Omega} \Big(d(\mathbf{x})^2 \Big)} \tag{9}$$

where $|\Omega|$ is the cardinality of the image domain. The standard RMS error gives an approximate average error for the entire signal, taking every pixel's error independently.

The second metric we use is the *normalised cross correlation* (NCC) percentage

$$C(s, s^*) = \frac{1}{|\Omega|} \sum_{\mathbf{x} \in \Omega} \frac{\big(s(\mathbf{x}) - \mu(s)\big)\,\big(s^*(\mathbf{x}) - \mu(s^*)\big)}{\sigma(s)\,\sigma(s^*)} \quad \times\ 100\% \tag{10}$$

where $\mu(h)$ and $\sigma(h)$ are the mean and standard deviation of image h, respectively. An NCC of 100 means that the images are (almost) identical, and an NCC of 0 means that the images have very large differences.

We calculated the NCC, $\max(d)$ and RMS for the entire dataset, the summary of results can be seen in the table below ($*$ is the don't-care character):

	Average	Minimum at (σ_1, ε)	Maximum at (σ_1, ε)
NCC (%)	100	99.994 at $(9, 10^{-4})$	100 at $(*, *)$
RMS (px)	8.7×10^{-5}	$< 10^{-12}$ at $(3, 10^{-12})$	0.0016 at $(9, 10^{-4})$
$\max(d)$ (px)	0.36	$< 10^{-12}$ at $(3, 10^{-12})$	5.3 at $(9, *)$

From this table it is very apparent that the fast-trilateral filter retains the smoothing properties of the LUT (original) version. The difference is only apparent when using high kernel values σ_1 and also low accuracy values (high ε). Even then, the errors are negligible. In fact, the maximum difference of any individual pixel was only 5.3, with an average maximum of 0.36 .

5 Conclusions and Future Research

In this paper we have covered the original implementation of the trilateral filter. We have suggested two speed improvements. One is using simple look-up-tables, and the other is using a novel way of truncating the look-up-table to a user specified required accuracy.

The speed improvements were shown to be drastic (in the order of several magnitudes) compared to the original or LUT trilateral filter. We identified that the fast-trilateral filter provides very accurate (almost identical) results, compared to the original (and LUT) trilateral filter. This massive speed gain for

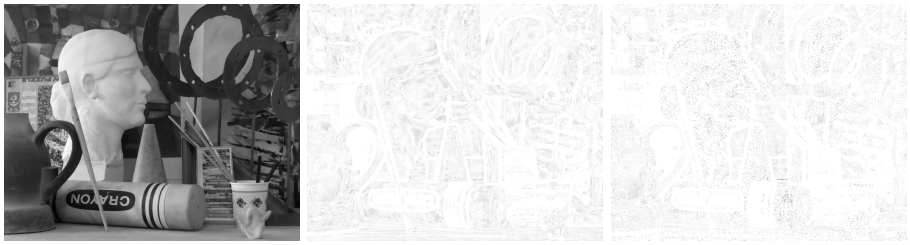

Fig. 4. Left: *Art* image that has been smoothed. Centre and Right: difference image between LUT and fast-trilateral filter, using $\varepsilon = 10^{-4}$ (centre, $\max(d) = 0.36$) and $\varepsilon = 10^{-12}$ (right, $\max(d) = 3.1 \times 10^{-5}$), with $\sigma_1 = 9$. Difference images scaled for visability, white $\leftrightarrow$ black into $0 \leftrightarrow \max(d)$.

a very small difference in results is a huge benefit, and thus makes the trilateral filter more usable.

Future work will be to improve speed using parallel architecture (e.g., GPU, Cell Processors, or OpenMP). Also, further applications of the trilateral filter have not been recognised as yet.

Acknowledgment: The authors would like to thank Prasun Choudhury (Adobe Systems, Inc., USA) and Jack Tumblin (EECS, Northwestern University, USA) for their original implementation of the trilateral filter.

References

1. Choudhury, P., and Tumblin, J.: The trilateral filter for high contrast images and meshes. In Proc. *Eurographics Symp. Rendering*, pages 1–11 (2003)
2. Gerig, G., Kubler, O., Kikinis, R., and Jolesz, F.A.: Nonlinear anisotropic filtering of MRI data. *IEEE Trans. Medical Imaging*, **11**:221–232 (2002)
3. Hirschmüller, H., and Scharstein, D.: Evaluation of stereo matching costs on images with radiometric differences. *IEEE Trans. Pattern Analysis Machine Intelligence*, to appear
4. Middlebury data set: stereo data http://vision.middlebury.edu/stereo/data/
5. Pharr, M., and Fernando, R.: *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*, Addison-Wesley, online (2005)
6. Tobi Vaudrey's homepage: http://www.cs.auckland.ac.nz/∼tobi/
7. Tomasi, C., and Manduchi, R.: Bilateral filtering for gray and color images. In Proc. *IEEE Int. Conf. Computer Vision*, pages 839–846 (1998)
8. Vaudrey, T., and Klette, R.: Residual images remove illumination artifacts for correspondence algorithms! Technical report, www.mi.auckland.ac.nz, The University of Auckland (2009).
9. Wong, W. C. K., Chung, A. C. S., and Yu S. C. H.: Trilateral filtering for biomedical images, In Proc. *IEEE Int. Symp. on Biomedical Imaging: From Nano to Macro (ISBI)*, pages 820–823 (2004)