# A Simplified High Dynamic Range Bilateral Filter

David Beynon

June 17, 2011

## 1 Abstract

An optimisation is presented which allows an efficient translation invariant implementation of the bilateral filter with a simplified weighting function which is suitable for use with high dynamic range images.

The algorithm relies on a reformulation of the bilateral filter, which allows some of the advantages of histogram based methods to be directly applied to high dynamic range data. In particular it owes a debt to the work of Weiss[4].

## 2 Introduction

The bilateral filter is a nonlinear edge preserving low pass filter in which the value of each pixel is replaced by a weighted sum of nearby pixels, with weights dependent on both the spatial distance and difference in value from the "central" pixel.

In the original form of the filter the Gaussian function $G(\sigma, x)$ is used for both the spatial and value terms. In a two dimensional image this would give us the weighting function:

$$\begin{aligned} W(\sigma_{spatial}, \sigma_{value}, x_0, y_0, value_0, x, y, value) &= G(\sigma_{spatial}, sqrt((x-x_0)^2 + (y-y_0)^2)) * \\ & \quad G(\sigma_{range}, value - value_0) \end{aligned}$$

The inclusion of the "value" term results in a blurring filter that does not propagate over strong edges, but does remove fine detail and noise from a signal.

The main drawback of the bilateral filter is its computational complexity. A naive implementation requires weights to be calculated for every pixel within the spatial extent of the filter. For a circular filter with radius $r$ this results in a a computational cost of $O(r^2)$, which can lead to computation times of hours for filter sizes on the order of $r = 100$.

More recent work has reduced the cost to $O(log(r))$[4] and even $O1$[3] for low dynamic range images, but the nature of histogram based methods makes it difficult to apply them to high dynamic range data efficiently.

Durand & Dorsey[1] and Paris & Durand[2] introduced approximations which allow very fast processing of images. These are suited to many applications where exact shift invariance is not essential.

# 3 The Algorithm

## 3.1 Mathematical Derivation

If we simplify the bilateral filter so that its spatial term is a box filter then given a current pixel value $v$ and a weighting function $w(v)$ then the output value for the pixel will be given by:

$$B(v, w) = \sum vw(v) / \sum w(v) \tag{1}$$

In the case of a bilateral filter, we may choose a weighting function such that the weight is zero if the difference between $v$ and the current value at the centre of the filter, $v_0$, is greater than a threshold $t$. This gives us:

$$B(v, w) = \sum_{v=v_0-t}^{v_0+t} vw(v) / \sum_{v=v_0-t}^{v_0+t} w(v) \tag{2}$$

Given a spatial box filter and a simple quadratic weighting function:

$$w(v) = 1 - ((v_0 - v)/t)^2 \tag{3}$$

The weighting function expands to:

$$\sum_{v=v_0-t}^{v_0+t} w(v) = (1 - ((v_0 - v_1)/t)^2) + (1 - ((v_0 - v_2)/t)^2) + ... \quad (4)$$

and the value * weight function expands to:

$$\sum_{v=v_0-t}^{v_0+t} vw(v) = v(1 - ((v_0 - v_1)/t)^2) + v(1 - ((v_0 - v_2)/t)^2) + ... \quad (5)$$

If we define $v' = \frac{v}{t}$ then we can simplify this to:

$$\sum_{v'=v'_0-1}^{v'_0+1} w'(v') = n(1 - {v'_0}^2) + 2v'_0 \sum_{v'=v'_0-1}^{v'_0+1} v' - \sum_{v'=v'_0-1}^{v'_0+1} v'^2 \quad (6)$$

where $n$ is the total number of pixels with non zero weight. For the $vw(v)$ terms we get:

$$\sum_{v'=v'_0-1}^{v'_0+1} v'w'(v') = (1 - {v'_0}^2) \sum_{v'=v'_0-1}^{v'_0+1} v' + 2v'_0 \sum_{v'=v'_0-1}^{v'_0+1} v'^2 - \sum_{v'=v'_0-1}^{v'_0+1} v'^3 \quad (7)$$

therefore:

$$B(v, w) = t \left( \frac{(1 - {v'_0}^2) \sum_{v'=v'_0-1}^{v'_0+1} v' + 2v'_0 \sum_{v'=v'_0-1}^{v'_0+1} v'^2 - \sum_{v'=v'_0-1}^{v'_0+1} v'^3}{n(1 - {v'_0}^2) + 2v'_0 \sum_{v'=v'_0-1}^{v'_0+1} v' - \sum_{v'=v'_0-1}^{v'_0+1} v'^2} \right) \quad (8)$$

### 3.2 Data structures

In order to make use of this derivation we need to be able to efficiently retrieve values of the $n$, $\sum_{v'=v'_0-t}^{v'_0+t} v'$, $\sum_{v'=v'_0-t}^{v'_0+t} v'^2$ and $\sum_{v'=v'_0-t}^{v'_0+t} v'^3$ terms.

The current implementation uses an AVL tree in which each node stores the values $v'$, $v'^2$, $v'^3$ and the sums of these values and number of nodes in its sub tree.

The distributive property of histograms exploited by Weiss[4] also applies to the totals used by this algorithm. This allows some of the same optimisations to be employed.

A multi level data structure spans the width of the image or tile to be processed. The finest level consists of a single AVL tree per column of pixels. Subsequent levels contain fewer trees, each of which covers a larger area of the image.

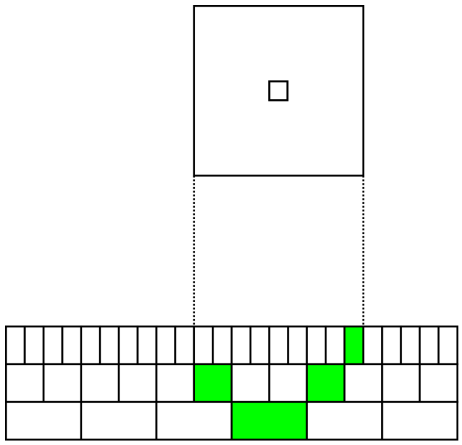

Figure 1: Multi level data structure. Trees accessed while calculating the filtered value of the pixel are coloured green

The structure is populated with the values of all pixels in the horizontal strip of image associated with scan lines in the range $y - r \leq y \leq y + r$. For each pixel in the current scan line the structure is traversed in such a way as to minimise the number of tree look ups. The structure is incrementally updated by removing pixels on the line $y - r$, incrementing $y$ and adding pixels on the line $y + r$.

### 3.3 Implementation details

The AVL tree data structure is much cheaper to access than it is to update, so it makes sense to perform some tuning of the data structure outlined in

figure 1. In our implementation we found that having the trees in each layer cover 4 times the number of pixels as the layer below was reasonable.

Limiting the number of layers so there was always one less than we would get if we allowed them to grow to the diameter of the filter also helped. The performance impact of changing this detail is shown in figure 5.

In order to avoid floating point precision issues we encode floating point values as integers when operating on $log(intensity)$ values. A signed 64-bit integer has sufficient precision to support filters up to around $r = 100$, or 400000 pixels with a precision of $10^{-6}$ for reasonable values of $t$.

Our implementation makes use of multiple processors or cores by splitting the image into tiles. In order to mitigate the cost of the initial setup of the data structures it is best to make the tiles themselves quite large. A default size of 512x1024 seems to work well for photographic use.

Early implementations suffered from serious scaling problems due to locking in the system memory allocator. The current version uses simple list based thread local allocators to avoid this problem. The results of scaling tests are shown in figures 6 and 7.

### 3.4 Possible extensions

The algorithm may be extended to any function of the form $w(v) = 1 - ((v_0 - v)/t)^n$ with positive integer values of $n$, simply by adding terms up to $n + 1$. Even values of $n$ are preferable, as an odd exponent requires values above and below $v_0$ to be handled separately, which introduces a performance penalty.

As the algorithm only depends on the sums of values within its range it is possible to implement a "bilateral filter" with an infinite spatial extent very efficiently. For each pixel create a record containing the pixels luminance and its original position in the image. These records are sorted in order of value, and traversed in order. 3 pointers are maintained, to the current pixel, and the minimum and maximum pixels to be considered.

The basic formulation also lends itself to filtering in any number of dimensions, although the design of suitable data structures for more than two is beyond the scope of this document.

# 4 Performance

All performance tests were performed on a 9.5 megapixel HDR image. Tests of our algorithm were performed by timing the hdrshop plug in `stm_bilateral_filter.exe` on a dual quad core Xeon running at 2.26GHz. All the tests were forced to run on a single core. Times include any file loading, saving and setup time performed by the applications.

Each test consisted of 3 passes over the image. The corresponding $\sigma$ value was calculated in order to allow easy comparison with other implementations.

Two other bilateral filter implementations were compared. The first was an exact implementation with SSE optimisation, which is available as part of Francesco Banterle's collection of hdrshop plugins. The second was an implementation of Sylvain Paris and Frédo Durand's fast bilateral filtering algorithm taken from their web site.

The comparison with the exact algorithm is shown in figure 2. The new algorithm shows a substantial improvement in performance, and appears to show near linear scaling with filter radius over the range tested, albeit with a large constant overhead. An exact analysis has not been carried out, but there is reason to believe it scales with $log^2(r)$.

Figure 3 shows a comparison with the "truncated kernel" version of Paris and Durand's 2006 algorithm. Due to the file loading code in their test program the test was carried out with a low dynamic range version of the image, but there is no reason to believe that performance with high dynamic range data would be any different. Performance of Paris and Durand's algorithm is extremely strong, and almost unaffected by filter size.

A comparison of all three approaches is shown in figure 4.

# 5 Results

Figure 8 demonstrates use of the filter on the $log(luminance)$ channel of a low dynamic range image, and the results of using it for contrast enhancement.

Figure9 demonstrates tone mapping an image using a global algorithm, and contrast enhancement with our algorithm as a preprocess.

## 6 Resources

An implementation of the algorithm can be found as part of the `STM` high dynamic range image processing toolkit, which is available at: `http://www.spectral3d.co.uk/Vapourware/tonemap/`.

The source code for the bilateral filter itself is in the directory: `http://www.spectral3d.co.uk/repo/stm/processors/bilateral`.

An implementation of the "infinite" bilateral filter can be found in the directory: `http://www.spectral3d.co.uk/repo/stm/processors/global`.

The SSE optimised exact implementation used for benchmarking is available as part of "Banty's Toolkit", which can be found in Francesco Bantarele's web site at: `http://www.banterle.com/francesco/`.

The paper and source code for the approximate bilateral filter is currently available on Sylvain Paris' project page here: `http://people.csail.mit.edu/sparis/bf/`.

## References


[1] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *ACM Transactions on Graphics*, volume 21(3) of *Proceedings of the SIGGRAPH conference*. ACM, 2002.

[2] Sylvain Paris and Frédo Durand. A fast approximation of the bilateral filter using a signal processing approach. In *In Proceedings of the European Conference on Computer Vision*, pages 568–580, 2006.

[3] Fatih Porkili. Constant time o(1) bilateral filtering. In *Computer Vision and Pattern Recognition*. IEEE, 2008.

[4] Ben Weiss. Fast median and bilateral filtering. In *ACM Transactions on Graphics*, volume 25(3) of *Proceedings of the SIGGRAPH conference*. ACM, 2006.

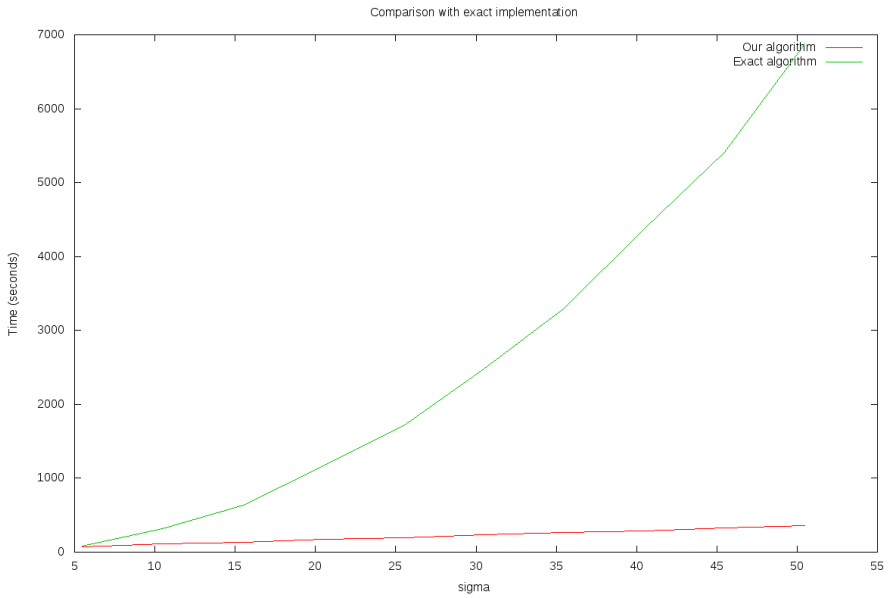


Figure 2: Comparison with exact SSE implementation

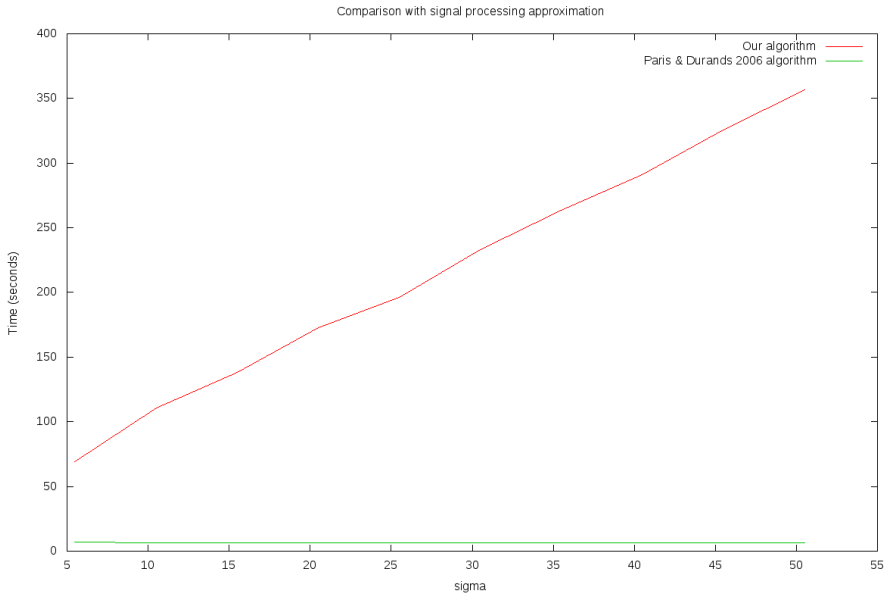


Figure 3: Comparison with Paris & Durand's approximate algorithm

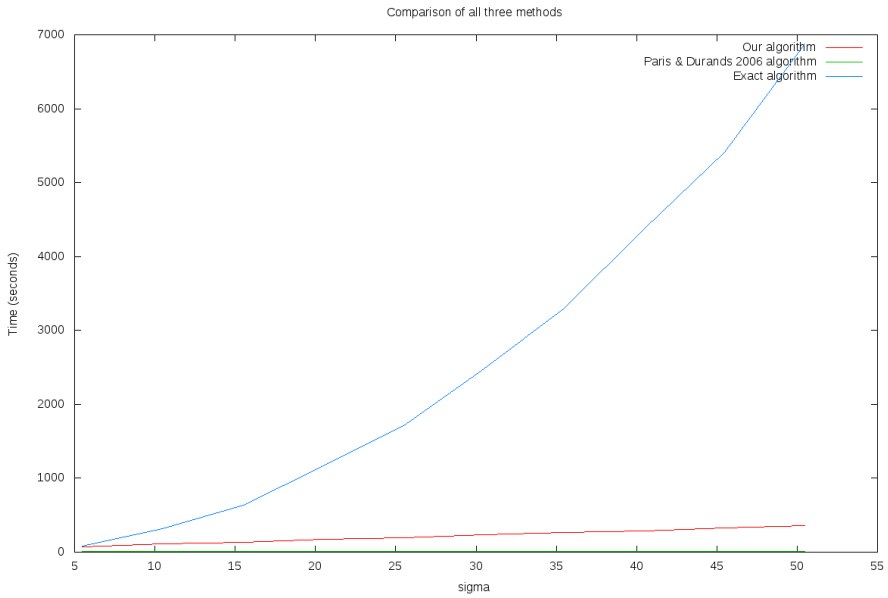


Figure 4: Comparison with exact algorithm and Paris & Durand's approximate algorithm

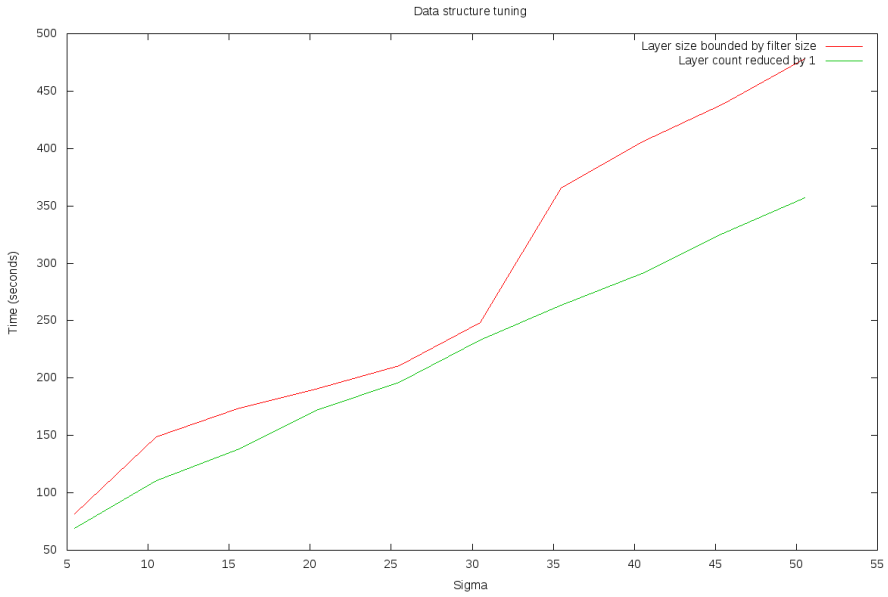


Figure 5: Effect of minor data structure tuning. Note the steps in the red plot when the layer count increases

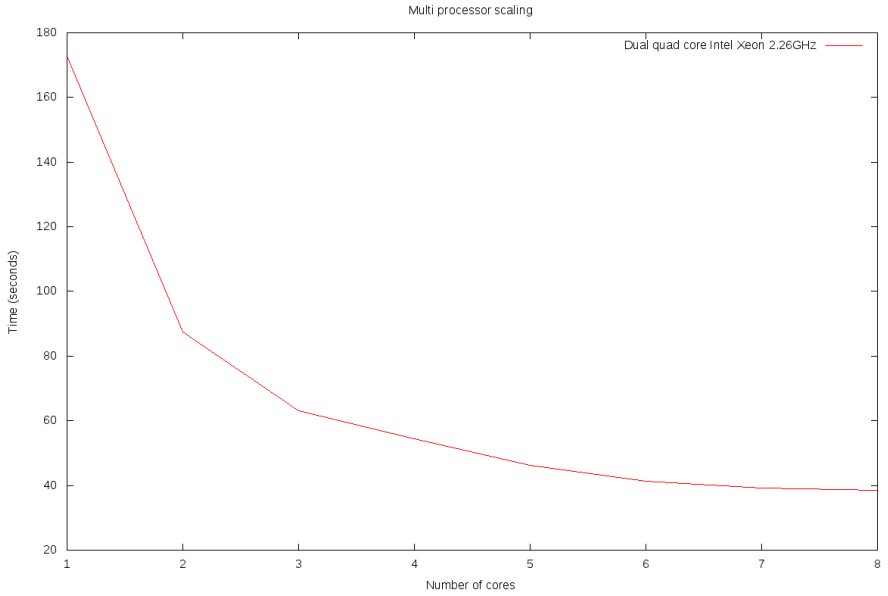


Figure 6: Scaling behaviour on a dual quad core Intel Xeon. 3 passes with a filter radius of 20

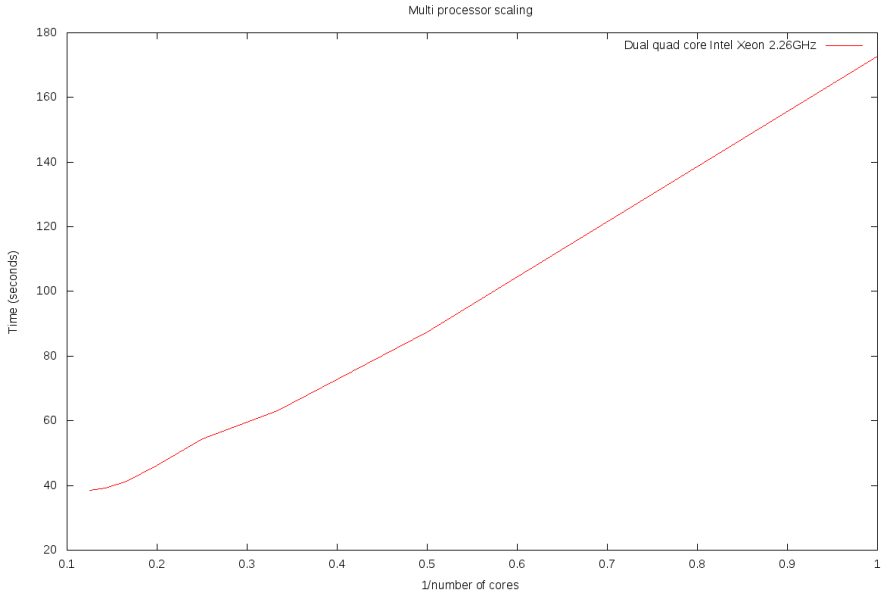


Figure 7: Scaling behaviour on a dual quad core Intel Xeon. 3 passes with a filter radius of 20

(a) Original (b) Bilateral filter (c) Contrast enhancement

Figure 8: Bilateral filtering and contrast adjustment of a low dynamic range image

(a) Global operator

(b) Local operator using bilateral filter

Figure 9: Tone mapping