Burst photography for high dynamic range and low-light imaging on mobile cameras

Supplemental Material

Samuel W. Hasinoff
Jonathan T. Barron
Dillon Sharlet
Florian Kainz
Ryan Geiss
Jiawen Chen
Andrew Adams
Marc Levoy
Google Research

1 Brute-force L1 Alignment

At the finest scale of our coarse-to-fine alignment strategy, we require an alignment technique which performs well given large tile sizes but a very small search radius. In this context, techniques such as our previously-described fast subpixel L2 alignment, or even simpler techniques such as phase correlation (Kuglin and Hines 1975) are outperformed by a well-implemented brute-force procedure which minimizes absolute residuals. We use absolute residuals instead of squared residuals because they map well to low level computer architectures. In contrast with squared residuals, absolute residuals require fewer bits for the same input data, enabling higher throughput. On ARM architectures, there is in fact an operation making it most appealing in contexts where the search radius is small.

2 Image Alignment Pyramids

The image pyramids we use for alignment are constructed to balance computational effort and the quality of the alignment results. The main control we have over quality versus computation is the computational effort and the quality of the alignment results. At the same scale of our coarse-to-fine alignment strategy, we reduce the tile size to $1 \times 16$. At the finest scale of our coarse-to-fine alignment strategy, we reduce the tile size to $1 \times 16$.

1 Brute-force L1 Alignment

At the finest scale of our coarse-to-fine alignment strategy, we require an alignment technique which performs well given large tile sizes but a very small search radius. In this context, techniques such as our previously-described fast subpixel L2 alignment, or even simpler techniques such as phase correlation (Kuglin and Hines 1975) are outperformed by a well-implemented brute-force procedure which minimizes absolute residuals. We use absolute residuals instead of squared residuals because they map well to low level computer architectures. In contrast with squared residuals, absolute residuals require fewer bits for the same input data, enabling higher throughput. On ARM architectures, there is in fact an operation making it most appealing in contexts where the search radius is small.

2 Image Alignment Pyramids

The image pyramids we use for alignment are constructed to balance computational effort and the quality of the alignment results. The main control we have over quality versus computation is the computational effort and the quality of the alignment results. At the same scale of our coarse-to-fine alignment strategy, we reduce the tile size to $1 \times 16$. At the finest scale of our coarse-to-fine alignment strategy, we reduce the tile size to $1 \times 16$.

1 Brute-force L1 Alignment

At the finest scale of our coarse-to-fine alignment strategy, we require an alignment technique which performs well given large tile sizes but a very small search radius. In this context, techniques such as our previously-described fast subpixel L2 alignment, or even simpler techniques such as phase correlation (Kuglin and Hines 1975) are outperformed by a well-implemented brute-force procedure which minimizes absolute residuals. We use absolute residuals instead of squared residuals because they map well to low level computer architectures. In contrast with squared residuals, absolute residuals require fewer bits for the same input data, enabling higher throughput. On ARM architectures, there is in fact an operation making it most appealing in contexts where the search radius is small.

2 Image Alignment Pyramids

The image pyramids we use for alignment are constructed to balance computational effort and the quality of the alignment results. The main control we have over quality versus computation is the computational effort and the quality of the alignment results. At the same scale of our coarse-to-fine alignment strategy, we reduce the tile size to $1 \times 16$. At the finest scale of our coarse-to-fine alignment strategy, we reduce the tile size to $1 \times 16$.

1 Brute-force L1 Alignment

At the finest scale of our coarse-to-fine alignment strategy, we require an alignment technique which performs well given large tile sizes but a very small search radius. In this context, techniques such as our previously-described fast subpixel L2 alignment, or even simpler techniques such as phase correlation (Kuglin and Hines 1975) are outperformed by a well-implemented brute-force procedure which minimizes absolute residuals. We use absolute residuals instead of squared residuals because they map well to low level computer architectures. In contrast with squared residuals, absolute residuals require fewer bits for the same input data, enabling higher throughput. On ARM architectures, there is in fact an operation making it most appealing in contexts where the search radius is small.

2 Image Alignment Pyramids

The image pyramids we use for alignment are constructed to balance computational effort and the quality of the alignment results. The main control we have over quality versus computation is the computational effort and the quality of the alignment results. At the same scale of our coarse-to-fine alignment strategy, we reduce the tile size to $1 \times 16$. At the finest scale of our coarse-to-fine alignment strategy, we reduce the tile size to $1 \times 16$.
Just as before, this distance calculation can be simplified:

\[ D_2(u, v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} T(x, y)^2 \]
\[ + \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} I(x + u, y + v)^2 \]
\[ - 2 \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} T(x, y)I(x + u, y + v) \]

The first term depends only on \( T \) and not at all on \( u \) and \( v \), and so it can be computed once and re-used when computing all values of \( D_2(u, v) \). The second term can be computed for all values of \( \{u, v\} \) by box filtering \( I(x, y)^2 \), which can be done efficient using sliding-window image filtering techniques or (somewhat less efficiently) using integral images. And the third term can also be computed for all values of \( \{u, v\} \) by cross-correlating \( I \) and \( T \). Cross-correlation can be expensive to compute naively, but can be sped up significantly using fast Fourier transforms. From the convolution theorem, we know that:

\[ a \ast b = F^{-1} \{ F\{a\}^\ast \circ F\{b\} \} \]

Where \( F\{\cdot\} \) is the Fourier transform, \( F^{-1}\{\cdot\} \) is the inverse Fourier transform, \( \circ \) is the pointwise product (Hadamard product) of two vectors, and \( F\{a\}^\ast \) is the conjugate transpose of \( F\{a\} \). This naturally generalizes from one-dimensional signals to two-dimensional images.

With these three observations, we can rewrite the computation of the distance “image” \( D_2 \) for all possible offsets \( (u, v) \) as:

\[ D_2 = \|T\|^2 + \text{box}(I \circ I, n) - 2 \left( F^{-1} \{\{I\}^\ast \circ F\{T\}\} \right) \]

Where the first term is the sum of the squared elements of \( T \), the second term is the squared elements of image \( I \) filtered with a box filter of size \( n \times n \) (where the box filter is not normalized), and the third term is \(-2\times\) the cross-correlation of \( I \) and \( T \), computed efficiently using the fast Fourier transform.

4 Subpixel Accurate Translation

Given distance image \( D_2(u, v) \), we would like to find the single best match between \( T \) and \( I \) by localizing the minimum of \( D_2 \). To produce subpixel quadratic function (a 2D polynomial), fit near the per-pixel minimum of \( D_2 \). This approach produces higher-quality translation estimates than the standard approach of fitting two separable functions [Stone et al. 2001], as jointly estimate the minimum in two dimensions rather than independently estimating each dimension. Minimum produces more accurate results in the case the axes of \( D_2(u, v) \) are not isotropic in \( u \) and \( v \), which they rarely are in practice (see figure 2c). More formally, we will approximate \( D_2 \) as follows:

\[ D_2(u, v) \approx \frac{1}{2} [u; v]^T A [u; v] + b^T [u; v] + c \]

Where \( A \) is a \( 2 \times 2 \) positive semi-definite matrix, \( b \) is a \( 2 \times 1 \) vector, and \( c \) is a scalar. \( A \) is assumed to be PSD because we expect the shape of \( D_2 \) near the minimum to be an upward-facing quadratic surface, rather than a saddle or a downward-facing surface. Let \( (u, v) \) be the coordinate of the pixel in \( D_2 \) with the smallest distance value. We will consider the \( 3 \times 3 \) pixel area around \((u, v)\) when fitting our quadratic function:

\[ D_2^{\text{sub}} = \begin{bmatrix} D_2(\hat{u}, \hat{v}) & D_2(\hat{u} + 1, \hat{v}) & D_2(\hat{u} + 1, \hat{v} + 1) \\ D_2(\hat{u} - 1, \hat{v}) & D_2(\hat{u}, \hat{v} + 1) & D_2(\hat{u} + 1, \hat{v} + 1) \\ D_2(\hat{u} - 1, \hat{v} + 1) & D_2(\hat{u} + 1, \hat{v}) & D_2(\hat{u} + 1, \hat{v} + 1) \end{bmatrix} \]

When fitting its bivariate polynomial, we will weight the pixels nearby the minimum according to a \( 3 \times 3 \)-sized patch of binomial weights:

\[ W = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \]

With \( D_2^{\text{sub}} \) and \( W \) we can set up a least-squares problem with respect to the free parameters in our quadratic approximation \((A, b, c)\) and solve it. Without loss of generality, we will solve for a fit of \( D_2^{\text{sub}} \) which assumes that the center pixel has a \((u, v)\) coordinate of \((0, 0)\), and then shift the sub-pixel position that we will estimate by \((\hat{u}, \hat{v})\). To construct our least-squares problem, we must first construct a
matrix $X$ which contains a second-order polynomial expansion of the 9 $(u, v)$ coordinates in our $3 \times 3$ patch:

$$X = \begin{bmatrix} \frac{1}{2} & 1 & \frac{1}{2} & -1 & -1 & 1 \\ 0 & \frac{1}{2} & 0 & -1 & 1 & 1 \\ \frac{1}{2} & -1 & \frac{1}{2} & 1 & -1 & 1 \\ \frac{1}{2} & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 1 & 0 & 1 \\ \frac{1}{2} & -1 & \frac{1}{2} & -1 & 1 & 1 \\ 0 & \frac{1}{2} & 0 & 1 & 1 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 1 & 1 & 1 \end{bmatrix}$$

(16)

We will additionally construct a diagonal “weight” matrix from our (vectorized) $3 \times 3$ weight “image” in equation 15, and a RHS vector $y$ as the vectorized version of $D_{2\text{sub}}$:

$$W = \text{diag}(\text{vec}(W))$$

(17)

$$y = \text{vec}(D_{2\text{sub}})$$

(18)

With these we can construct a least-squares problem which corresponds to fitting our bivariate polynomial:

$$\arg \min_\beta \left\| W^T(y - X\beta) \right\|^2$$

(19)

This is a conventional weighted least-squares problem, and so can be solved in a variety of ways. Ideally we would like to avoid repeatedly solving this linear system of equations for each tile in our alignment. This repeated fitting can be sped up significantly by taking advantage of the fact that $X$ and $W$ are constant across all tiles. We can therefore rearrange our linear system such that the polynomial fit parameters $\beta$ are a linear function of the image patch $y$ and a fixed matrix $F$:

$$\beta = F y$$

(20)

$$F = (X^T WX)^{-1} X^T W$$

(21)

From this we can see that we can compute the parameters of the bivariate polynomial by simply taking the inner product of $y$ (the $3 \times 3$ image patch of $D_2$, vectorized) by each row of a $F$. This matrix $F$ is equivalent to a filter bank, where each filter corresponds to some unknown parameter in $(A, b, c)$:

$$F_{A_{1,1}} = \begin{bmatrix} 1 & -2 & 1 \\ 2 & -4 & 2 \\ 1 & -2 & 1 \end{bmatrix} / 4, \quad F_{A_{2,2}} = \begin{bmatrix} 1 & 2 & -1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{bmatrix} / 4$$

$$F_{A_{1,2}} = F_{A_{2,1}} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} / 4$$

$$F_{b_1} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} / 8, \quad F_{b_2} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} / 8$$

$$F_c = \begin{bmatrix} -1 & 2 & -1 \\ 2 & 12 & 2 \\ -1 & -2 & -1 \end{bmatrix} / 16$$

(22)

With these filters we can estimate the free parameters of our quadratic approximation by simply taking the inner product of $D_{2\text{sub}}$ with these filters (assuming the error surface and the filter have been vectorized), or equivalently by computing the cross-correlation of $D_{2\text{sub}}$ with these filters:

$$A = \begin{bmatrix} F_{A_{1,1}} \cdot D_{2\text{sub}} & F_{A_{1,2}} \cdot D_{2\text{sub}} \\ F_{A_{2,1}} \cdot D_{2\text{sub}} & F_{A_{2,2}} \cdot D_{2\text{sub}} \end{bmatrix}$$

(23)

$$b = \begin{bmatrix} F_{b_1} \cdot D_{2\text{sub}} \\ F_{b_2} \cdot D_{2\text{sub}} \end{bmatrix}$$

(24)

$$c = F_c \cdot D_{2\text{sub}}$$

(25)

This process is similar to the polynomial expansion approach of [Farnebäck 2002]. The constant shift $c$ and its filter $F_c$ are irrelevant for our subpixel minimum localization, but are included here for the sake of completeness.

Depending on the shape of $D_{2\text{sub}}$, the estimated $A$ may not be positive semi-definite, contrary to our initial assumptions. To fix this, after estimating the parameters of our quadratic, we first force the diagonal elements of $A$ to be non-negative:

$$A_{1,1} \leftarrow \max(0, A_{1,1})$$

(26)

$$A_{2,2} \leftarrow \max(0, A_{2,2})$$

(27)

We then compute the determinant of $A$:

$$\det(A) = A_{1,1} A_{2,2} - A_{1,2}^2$$

(28)

if $\det(A) < 0$, then we set the off-diagonal elements of $A$ to be zero. These corrections give us a $A$ which is guaranteed to be positive semi-definite.

With our quadratic approximation, we can now estimate the minimum of that quadratic. Doing so requires that we rewrite our quadratic in a different form by completing the square:

$$\frac{1}{2} x^T A x + b^T x + c = \frac{1}{2} (x - \mu)^T A (x - \mu) + s$$

(29)

Given a quadratic defined in the first form, we can convert it into the second form as follows:

$$\mu = -A^{-1} b$$

(30)

$$s = c - \frac{\mu^T A \mu}{2}$$

(31)

For our particular bivariate case, this is equivalent to:

$$\mu = -\frac{[A_{2,2} b_1 - A_{1,2} b_2, A_{1,1} b_2 - A_{1,2} b_1]^T}{A_{1,1} A_{2,2} - A_{1,2}^2}$$

(32)

$$s = c - \frac{A_{1,1} \mu_1^2 + 2 A_{1,2} \mu_1 \mu_2 + A_{2,2} \mu_2^2}{2}$$

(33)

Once we have recovered the location of the minimum of the quadratic $\mu$, we can simply take that as the sub-pixel location of the minimum. Note that the fitted surface we produce treats the center pixel of $D_{2\text{sub}}$ as $(0, 0)$, so after fitting we need to add the per-pixel minimum location $(\hat{u}, \hat{v})$ into $\mu$, which gives us the actual location of the minimum in $D_2$. In the presence of severe noise or very flat images, it is possible for the predicted sub-pixel minimum $\mu$ to be very different from the observed per-pixel minimum $(\hat{u}, \hat{v})$, so in practice if we observe that the two are sufficiently different (more than 1 pixel removed) we set $\mu = [\hat{u}; \hat{v}]$.

5 Comparison with JPEG burst fusion

In our system, a key design decision is to use raw images as input to our align and merge algorithms, and then finish the raw merged result. Using raw images gives us both increased dynamic range and...
the ability to model sensor noise simply and accurately. By contrast, most previous burst fusion methods, e.g. [Liu et al. 2014; Dabov et al. 2007; Maggioni et al. 2012], consume JPEG images, which have already been finished by a photographic imaging pipeline.

To compare our system with such JPEG-based methods, we start from a dataset of 30 raw bursts and apply the same raw-to-JPEG finishing pipeline for all methods. For our method, this means running align and merge on raw bursts as usual, but substituting a different finishing pipeline. For JPEG-based methods, this means generating the JPEG input from the raw image bursts using the given finishing pipeline. This experimental approach lets us focus on the performance of the align and merge algorithms, without the confounding effect of the finishing pipeline, which can vary widely in tuning and overall quality across implementations.

**Experimental details** The 30-burst dataset we use for this evaluation is a subset of our larger dataset of several thousand raw bursts, to be released on publication, and includes the 10 bursts corresponding to figures 3-11. These bursts were captured for their coverage of different types of scenes, levels of motion, and brightness. The bursts were captured with 3 types of cameras, whose raw images are 12-13 Mpix.

For a raw-to-JPEG converter we used dcraw [Coffin 2016], followed by JPEG encoding at quality level 98, which effectively eliminates artifacts due to compression. While the pipeline implemented by dcraw is basic compared to commercial systems like Adobe Camera Raw, its predictability and lack of local tonemapping is an advantage for analysis. Furthermore, the AHD demosaicking method [Hirakawa and Parks 2005] implemented by dcraw works reasonably well in practice and is representative of the algorithms used by mobile ISPs. Color rendition in the results is somewhat compromised, due to limitations of both the DNG format and dcraw’s treatment of color metadata, but the effect is uniform across methods. Also note that some bursts are underexposed. This follows from our capture strategy for HDR scenes, together with the conservative global tonemapping applied by dcraw, which sets the white level at the 99th percentile.

We compare our method against several state of the art JPEG-based burst fusion methods from the academic literature: two variants of the burst denoising method proposed by Liu et al. [2014], as well as CV-BM3D [Dabov et al. 2007]. For [Liu et al. 2014], the authors used their implementation to process our dataset, holding settings fixed for all results. For 3 manually selected bursts, the authors brightened the input using a global tonemapping curve, consistent with the approach proposed in [Liu et al. 2014] for handling “extreme low-light” scenes. For CV-BM3D, we ran the authors’ Matlab implementation from the BM3D webpage.

Because this method does not include a mechanism for automatically setting the key noise level parameter, we ran a grid search over 17 different noise levels, residual wavelet basis functions were sometimes visible at a pixel scale in the result, and isolated hot pixels were sometimes visually exaggerated by the denoising.

To illustrate the performance of commercially available tools, we also compare against the JPEG-based “Merge to HDR Pro” feature of Adobe Photoshop CC 2015.1.2 [Adobe Inc. 2016] with “ghost removal” enabled, without further tonemapping. Although this Photoshop feature supports merging raw images as well, we found the HDR output unsuitable for input to dcraw because it already partially has photographic processing applied. In our experiments, Photoshop’s JPEG-based and raw-based results were qualitatively similar, so we only include the JPEG-based results in this comparison. We also tried the “Photo Merge HDR” feature in Lightroom CC 2015, but we found that when the input images all have the same exposure, this feature has no denoising effect; each pixel in the output is apparently derived from a single input frame.

**Summary of burst fusion results** We include full-resolution image results for all methods over all 30 bursts in supplemental material, to allow detailed inspection at 1:1 magnification. Here we summarize our high-level findings, and in figures 3–5 present results several illustrative bursts. Crops in these figures are roughly $600 \times 600$, so we encourage the reader to zoom in aggressively (300% or more) to appreciate fine pixel-level differences.

- In general, all the methods we evaluated are capable at handling the smooth motion due to camera shake for reasonably bright scenes. With moving subjects, more complicated occlusion relationships, or lower-light scenes, performance begins to degrade.
- We found Photoshop’s merging feature to be the most conservative of all methods, only implementing a very limited amount of denoising. Photoshop’s most notable artifact is strongly colored ghosts in regions of clipped pixels. It also sometimes produces thin “echoes” at boundaries of heavy motion.
- Both variants of [Liu et al. 2014] show artifacts at motion boundaries, where differing amounts of merging leads to discontinuities in the level of retained noise. Both of these methods occasionally demonstrate ghosting artifacts as well. In certain scenes, we also found that the fast pixel-based variant of [Liu et al. 2014] also shows a significant loss of contrast, possibly due to issues in pyramid blending.
- CV-BM3D behaves robustly with respect to motion across the 30-burst dataset, producing typical wavelet denoising results, without any artifacts that can be definitively ascribed to motion. Depending on the noise level chosen, results may look either too noisy or oversmoothed, but a reasonable balance was generally available, at the cost of some detail. For higher noise levels, residual wavelet basis functions were sometimes visible at a pixel scale in the result, and isolated hot pixels were sometimes visually exaggerated by the denoising.
- Our align and merge method, like CV-BM3D, is very robust to motion, with no objectionable artifacts across the 30-burst dataset. When alignment does break down, our method degrades gracefully to the base frame and the resulting denoising sometimes has the appearance of motion blur. Our method generally dominates all other approaches in this comparison at both detail preservation and denoising. We attribute this success primarily to our robust merging approach and the accurate noise model enabled by processing raw images.

As a reminder, this evaluation is only a comparison of alignment and merging quality. Our paper represents an entire system for both low-light and HDR imaging, from capture strategy to finishing, which runs efficiently on mobile devices and reliably produces artifact-free results.

**Runtime performance** As table 1 shows, performance for these burst fusion methods vary widely over several orders of magnitude. While platform differences make comparing runtimes challenging, it is clear that both our method and the faster pixel-based variant of [Liu et al. 2014] are at least an order of magnitude faster than all other methods in the comparison. After adjusting for platform differences, our method and the pixel-based variant of [Liu et al. 2014] still have roughly comparable performance. However, since
Figure 3: Burst fusion results, for a low-light scene with moderate motion. For readability the crops have been made uniformly brighter. Readers are encouraged to zoom aggressively (300% or more). Our method denoises effectively while retaining the finest detail of all methods. In areas where alignment was unsuccessful (foreground person in rightmost crop), our results degrade to the appearance of motion blur. CV-BM3D recovers less detail and produces a slightly blotchy appearance, but behaves robustly with motion. Photoshop has very little denoising effect, likely due to overly conservative deghosting. Both variants of [Liu et al. 2014] demonstrate ghosting (face in the middle crops) and show discontinuities in the amount of denoising near motion boundaries (rightmost crop).
Figure 4: Burst fusion results, for an indoor scene with heavy motion. Readers are encouraged to zoom aggressively (300% or more). Our method denoises while preserving detail, and shows no merging artifacts despite heavy motion and blurry input. CV-BM3D performs comparably but retains somewhat more noise. Photoshop has very little denoising effect, likely due to overly conservative deghosting. The [Liu et al. 2014] results take most image content from a different and sharper frame, however the fused result is oversmoothed, shows severe posterization and blocky artifacts (face and foot, two leftmost crops), and also demonstrates ghosting (flesh tone over shirt in leftmost crop, orange wood texture over boot in rightmost crop).
Figure 5: Burst fusion results, for a bright outdoor scene with varying motion. Readers are encouraged to zoom aggressively (300% or more). For this bright and relatively low dynamic range scene, merging confers limited improvement over capturing a single input frame. Our method, CV-BM3D, and Photoshop perform comparably, with the denoising effect most visible in low-texture regions. However, Photoshop introduces strong colored ghosting artifacts in clipped pixel regions (middle crop). Both variants of [Liu et al. 2014] demonstrate blocky artifacts (near left man’s back, leftmost crop) and sacrifice more fine detail than other methods. The pixel-based variant also produces hazy results for this burst (all crops), perhaps related to the pyramid blending approach.
Table 1: Align and merge runtime performance, averaged over a dataset of 30 bursts. The dataset consists of 264 images in total, each of which is 12–13 Mpix, corresponding to an average of 113 Mpix per burst.

<table>
<thead>
<tr>
<th>method (align and merge)</th>
<th>platform</th>
<th>type</th>
<th>cores used</th>
<th>average processing time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ours (align and merge)</td>
<td>Qualcomm Snapdragon 810</td>
<td>mobile</td>
<td>4+4 CPU</td>
<td>1.7</td>
</tr>
<tr>
<td>[Liu et al. 2014], pixel-based</td>
<td>i5 3.2GHz</td>
<td>desktop</td>
<td>1 CPU</td>
<td>2.2</td>
</tr>
<tr>
<td>[Liu et al. 2014], patch-based</td>
<td>i5 3.2GHz</td>
<td>desktop</td>
<td>1 CPU</td>
<td>40.7</td>
</tr>
<tr>
<td>CV-BM3D [Dabov et al. 2007]</td>
<td>i7 3.2GHz</td>
<td>desktop</td>
<td>1 CPU</td>
<td>300</td>
</tr>
<tr>
<td>Photoshop “Merge to HDR Pro”</td>
<td>i7 2.8GHz Macbook Pro</td>
<td>laptop</td>
<td>unknown</td>
<td>25</td>
</tr>
</tbody>
</table>

their implementation does not make use of SIMD they may have significant room for optimization.

References


MAGGIONI, M., BORACCHI, G., FOI, A., AND EGIAZARIAN, K. 2012. Video denoising, deblocking, and enhancement through separable 4-D nonlocal spatiotemporal transforms. TIP.