An Analysis and Implementation of the HDR+ Burst Denoising Method

Antoine Monod[1,2], Julie Delon[1], Thomas Veit[2]

[1] MAP5, Université de Paris, France ({antoine.monod, julie.delon}@u-paris.fr)
[2] GoPro Technology, France (tveit@gopro.com)

Abstract

HDR+ is an image processing pipeline presented by Google in 2016. At its core lies a denoising algorithm that uses a burst of raw images to produce a single higher quality image. Since it is designed as a versatile solution for smartphone cameras, it does not necessarily aim for the maximization of standard denoising metrics, but rather for the production of natural, visually pleasing images. In this article, we specifically discuss and analyze the HDR+ burst denoising algorithm architecture and the impact of its various parameters. With this publication, we provide an open source Python implementation of the algorithm, along with an interactive demo.

Source Code

The Python implementation of HDR+ has been peer-reviewed and accepted by IPOL. The source code, its documentation, and the online demo are available from the web page of this article[1]. Compilation and usage instructions are included in the README.txt file of the archive. The code is also available on GitHub[2].

Keywords: burst denoising; computational photography; motion estimation; temporal filtering; high dynamic range; image signal processor

1 Introduction

Image noise is a common issue in digital photography, and that issue is even more prominent in smaller sensors used in devices like smartphones or action cameras. Continuous improvement in the processing power of small devices has enabled many advances in computational photography, where algorithms are used to compensate for the limited physical properties of the imaging system. Algorithms that aim at removing digital noise are a very recurrent topic of the computational photography literature. Most of them only use the information stored in the input image (they are usually referred to as "single-frame" algorithms), while others suggest the use of information stored

---

[1] https://doi.org/10.5201/ipol.2021.336
[2] https://github.com/amonod/hdrplus-python

in additional images (usually videos or bursts). These multi-frame "denoisers" usually make use of similarity measurements in order to combine the information of multiple pixels or patches [3].

A multi-frame denoising algorithm is at the heart of the HDR+ system. It was first officially introduced as a feature within the Google Camera App in the newly released Nexus 5 and Nexus 6 smartphones in fall 2014[3]. In 2016, a full raw image processing pipeline with the same HDR+ moniker was presented in [9]. This method is the main subject of this article. The pipeline was embedded in smartphones such as the Google Nexus 6, 5X and 6P and first generation Google Pixel. Further algorithm refinements and optimizations were included in the Google Pixel 2 which was released in fall 2017. This device was the first to embed HDR+ in a dedicated Google-designed Image Processing Unit[4]. The same raw burst alignment and merging technique was included in a new "Night Sight" mode that was launched alongside the Pixel 3A. The changes and optimizations of this mode were presented in a new publication in 2019 [15]. As such, the complex HDR+ system is not only part of a detailed, frequently compared and quoted scientific publication, but also embedded into multiple recent mass-produced consumer electronics devices (which have received praise for their camera performance[5]), making it a relevant subject of study.

In this article, we will describe the core part of the system, a raw burst denoising algorithm (although it will not be our main focus, we will also mention the rest of the pipeline). This publication also comes with a publicly available HDR+ Python implementation (with a simplified finishing pipeline), along with an interactive demo on IPOL.

2   Characteristics of the HDR+ Pipeline

HDR+ was designed as a consumer-centric smartphone photography pipeline. Its goal is to produce individual pictures with good contrast and dynamic range, little noise and motion blur, and pleasing colors in most shooting scenarios, all while looking natural and requiring little to no user input. In order to achieve that goal, the authors of [9] conceived a system with the following constraints:

- The images that are captured and processed are raw images, as they typically have higher bit depth and a better known noise model than their post-processed and compressed 8-bit JPEG counterparts.

- Images are underexposed at capture time to avoid clipped highlights and saturated pixels, which effectively allows the capture of more dynamic range. Thanks to a custom auto-exposure algorithm, a gain is also memorized to compensate later in the pipeline for that under-exposure.

- Images also have high shutter speed (or short exposure time) to avoid motion blur (from the scene or the camera).

- For a fixed gain, images that have a shorter exposure time tend to have a lower signal-to-noise ratio; to compensate for that, multiple images are actually captured (usually between 2 and 8 images depending on scene brightness measurements, also selected through the auto-exposure algorithm), and the temporal information of the burst is combined through an alignment and merging procedure. All images have identical shutter speed (meaning that unlike some HDR fusion methods, exposure is not bracketed) and the interval between each frame is constant.

---

[3]HDR+: Low Light and High Dynamic Range photography in the Google Camera App, https://ai.googleblog.com/2014/10/hdr-low-light-and-high-dynamic-range.html, 2014

[4]Pixel Visual Core: image processing and machine learning on Pixel 2, https://www.blog.google/products/pixel/pixel-visual-core-image-processing-and-machine-learning-pixel-2/

[5]https://www.dxomark.com/google-pixel-2-reviewed-sets-new-record-smartphone-camera-quality/

This allows both a similar noise profile across the burst and easier, more accurate motion estimation and compensation.

- The resulting higher bit depth, high dynamic range, and less noisy image is then tone mapped to compensate for underexposure and to produce a more natural look, even in high contrast scenarios (in that case, shadows are boosted while as few highlights as possible are clipped).

- Additional processing is performed to produce a more visually pleasing final image with a distinctive "HDR+ look" (c.f. Section 5.1).

- Total burst processing time on a mobile device is only a few seconds, and the mode is transparent to the user (they do not know that several pictures are being taken or what processing needs to be performed).

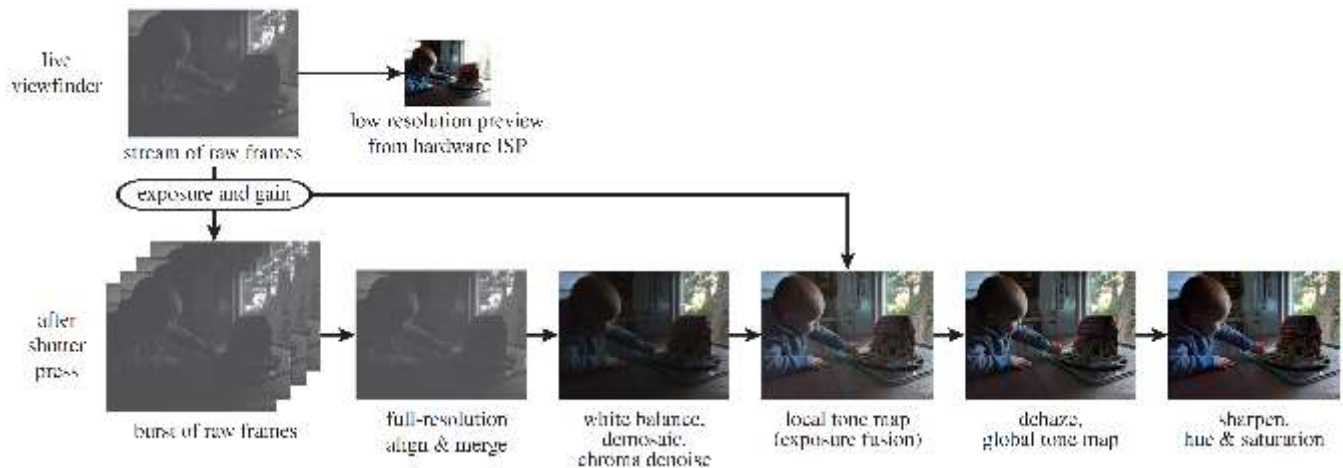- The whole process is automatic and parameter-free for the photographer.

Figure 1: The HDR+ pipeline (extracted from [9], lens shading and chromatic aberration correction have been omitted for brevity)

A representation of the whole image processing pipeline can be seen in Figure 1. For more information on the capture process, including the functioning of the example-based auto-exposure algorithm, please refer to the original article [9], and its supplemental material[6]. In the next sections, we will explain the whole processing pipeline after capture.

3 Bayer Tile-based Alignment

Before combining the information of multiple noisy images to produce a single, clean image, one must make sure that all images feature similar content. Since they share the same exposure time, scene brightness can be considered constant for the duration of the burst. Even though that exposure time is short, there can be apparent motion in the burst, mainly due to camera shake (the pipeline being designed for smartphones, most pictures will be taken handheld) and objects moving in the scene. That motion has to be compensated.

The strategy to compensate motion across the burst is to cleverly select one image as a reference, and then estimate motion between that image and each one of the other burst images (sometimes

---

[6]https://static.googleusercontent.com/media/hdrplusdata.org/en//hdrplus_supp.pdf

referred to as alternate frames). To that end, the reference image is divided into a regular array of square tiles and, for each tile, the corresponding tile in the alternate image is the one of minimal distance within a defined search radius around the reference tile's original location. Estimating the motion of image tiles instead of individual pixels is sufficient for the merging phase, and as we will explain in Section 4, the rest of the algorithm is robust to alignment errors.

3.1 Reference Image Selection

The reference image is selected among the first three images of the burst in order to minimize perceived shutter lag (the content of subsequent images of the burst can be quite different from what it was when the user pressed the shutter). Among these, the sharpest image is picked, the reasoning being that images that are less sharp are more likely to feature motion blur, and will not merge well with images that are sharp in the same regions. Sharpness is measured by the magnitude of gradients within a single (green) channel. We did not re-implement this part of the algorithm in our own implementation: for bursts provided by Google (which we'll discuss in Section 6.1), we simply reuse their selected reference image. In the case of new bursts, we let the user select the reference or default to the first image otherwise.

For the remainder of the alignment step, the full resolution Bayer images are converted to single channel, lower resolution grayscale images using a simple 2×2 box filter: each R, G, G, B square Bayer pattern is averaged in order to produce a single intensity value. Since the motion estimation of the grayscale images produces final results at pixel level, motion can only be estimated in multiples of 2 pixels in the original full-size raw images. Although this could be considered as a lack of precision, it is actually convenient because the arrangement of color planes in the Bayer image will be preserved after alignment (the value of a blue pixel will never be replaced by the value of a red pixel).

3.2 Multi-scale Pyramid Alignment

In order to quickly cover a large search area to find the tiles with minimal distance, the following multi-scale coarse-to-fine alignment strategy is adopted:

- Gaussian pyramids of the single channel burst images are constructed (e.g. in the HDR+ article supplement, it is typically 4-level, with successive downsampling factors of 2, 4 and 4 from the grayscale image to the coarsest level of the pyramid).

- At each pyramid level starting from the coarsest (i.e. lowest resolution) level, the reference image is divided in equally spaced square tiles (e.g. in the HDR+ supplement tiles of size 8×8 at the coarsest level and 16×16 at other levels). For each reference tile, we compute the L2 or L1 distance between it and each possible tile of the alternate frame within a specified search radius (e.g. + or - 4 pixels) around some initial guess (that guess being the location of the reference tile at the coarsest level)

$$D_p(u, v) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} |T(i, j) - I(i + u + u_0, j + v + v_0)|^p, \tag{1}$$

where T is the reference tile of size $n \times n$, (u, v) is the possible location of the alternate image tile within the larger search area I currently evaluated, p is the power of the norm used (1 or 2) and (u_0, v_0) is the initial guess which indicates the location of the search area ($u_0 = v_0 = 0$ at the coarsest scale).

As suggested in the HDR+ supplement, the L2 distance (written in the case where $u_0 = v_0 = 0$ for brevity) can be rewritten as

$$D_2(u, v) = \sum_{x=0}^{n-1}\sum_{y=0}^{n-1} T(i, j)^2 + \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} I(i+u, j+v)^2 - 2\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} T(i, j)I(i+u, j+v), \quad (2)$$

where the second term can be computed by filtering the squared elements of I with a $n \times n$ box filter, and the third is proportional to the cross-correlation of I and T, which can be computed quickly using fast Fourier transforms.

The displacement from the location of the reference tile to the location of the alternate tile with minimal distance gives us the motion vector attributed to the reference tile.

- Alignments are upsampled, meaning that motion vectors are scaled and propagated to the correct number of tiles in the finer level. For example, if tiles are 16×16 at both levels and the upsampling factor is 4, a tile with a (2, 1) motion vector at the coarser level implies 16 tiles with a (8, 4) motion vector at the finer level.

- An additional step is added in order to mitigate some potential upsampling problems, particularly when coarse-scale tiles straddle over the boundaries of moving objects. For each tile at a new (finer) level, instead of directly using the upsampled motion vector of the corresponding coarse-scale tile as the initial alignment guess (u_0, v_0), 3 candidates are evaluated: the alignment of the coarse-scale tile, plus those of the nearest coarse-scale tiles in each spatial dimension. The candidate alignment that minimizes the L1 distance between the reference tile and the corresponding alternate tile at the new pyramid level becomes the new initial alignment guess. This step is illustrated in Figure 2.
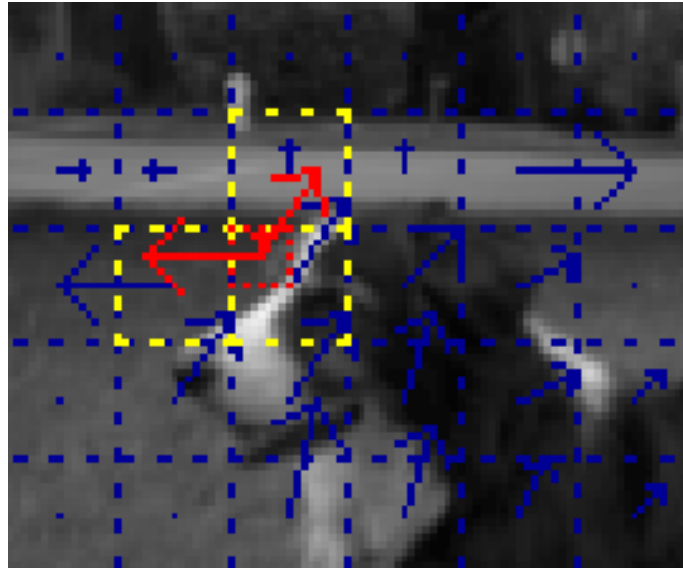

Figure 2: **Blue**: coarse-scale motion field (arrows: motion vectors, dotted lines: tiles). **Red**: finer-scale tile. **Yellow**: its 3 nearest coarse-scale tiles. The 3 motion vectors extracted from those neighboring tiles will be the candidates evaluated at the finer scale (shown in **red**).

- The selected upsampled motion vectors are then used as the location of the search area for the motion estimation at the subsequent pyramid level. For example, if (u_m, v_m) is the upsampled motion vector resulting from the previous steps, the initial guess (u_0, v_0) will be updated to $(u_0 + u_m, v_0 + v_m)$.

(a) Image pair (ref=top) (b) Coarse-to-fine motion fields (hue: direction, saturation: magnitude)
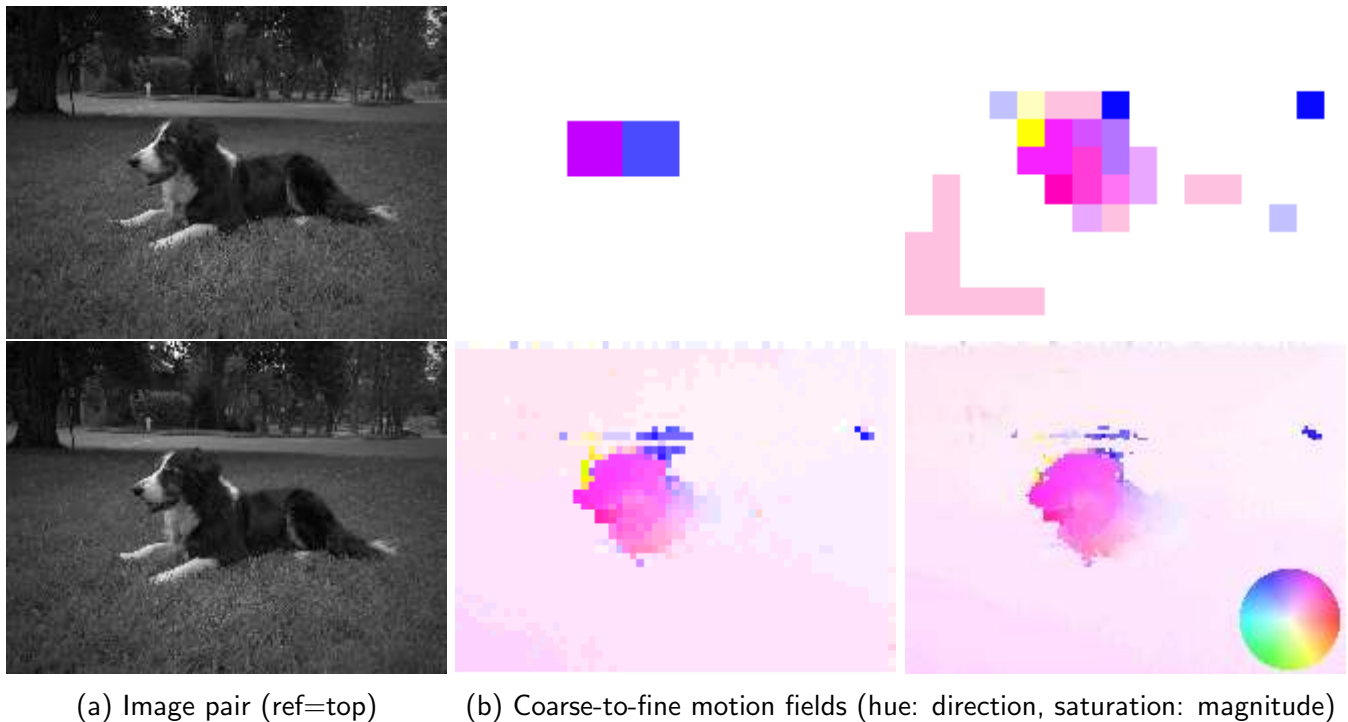
Figure 3: Multi-scale tile-based alignment using Gaussian pyramids

An example of coarse-to-fine motion fields can be seen in Figure 3. Given that the alignments are computed for downsampled grayscale versions of the original images, motion vectors must be multiplied by a factor of 2 to go back to the raw Bayer array size.

3.3 Subpixel Alignment

We implemented subpixel alignment as defined in the article: between two pyramid scales, for each motion estimation result, we fit a bivariate quadratic polynomial to the 3×3 window surrounding the L2 distance minimum, and find the minimum of the polynomial.

As described in the article supplement[7], D_2 can be approximated as

$$D_2(u,v) \approx \frac{1}{2} [u;v]^\top \mathbf{A} [u;v] + \mathbf{b}^\top [u;v] + c, \tag{3}$$

where $\mathbf{A}$ is a 2×2 positive semi-definite matrix (because the shape of D_2 is assumed to be an upward-facing quadratic surface near the minimum), $\mathbf{b}$ is a 2×1 vector and c is a scalar.

If $(\hat{u}, \hat{v})$ is the estimated distance minimum, the technique consists in constructing a weighted least-squares problem fitting a quadratic polynomial to the 3×3 window of D_2 centered around $(\hat{u}, \hat{v})$, which we call D_2^{sub}. Without loss of generality, we can solve the problem assuming $(\hat{u}, \hat{v}) = (0,0)$ and then shift the estimated subpixel position μ by $(\hat{u}, \hat{v})$.

The free parameters of the quadratic can be estimated by computing the inner product of D_2^{sub}

---

[7]https://static.googleusercontent.com/media/hdrplusdata.org/en//hdrplus_supp.pdf

with a set of six 3×3 filters, each corresponding to an unknown parameter in $(\mathbf{A}, \mathbf{b}, c)$

$$
\begin{aligned}
\mathbf{A} &= \left[\begin{array}{cc} F_{A_{1,1}} \cdot D_2^{sub} & F_{A_{1,2}} \cdot D_2^{sub} \\ F_{A_{1,2}} \cdot D_2^{sub} & F_{A_{2,2}} \cdot D_2^{sub} \end{array} \right], \\
\mathbf{b} &= \left[\begin{array}{c} F_{b_1} \cdot D_2^{sub} \\ F_{b_2} \cdot D_2^{sub} \end{array} \right], \\
c &= F_c \cdot D_2^{sub}.
\end{aligned}
\tag{4}
$$

In some cases, additional operations might be required so that $\mathbf{A}$ is guaranteed positive semi-definite. Please refer to the supplement for demonstrations, including the derivation of the filters.

Once the quadratic approximation parameters are estimated, we can compute the minimum of the fitted surface

$$
\mu = -\mathbf{A}^{-1}\mathbf{b},
\tag{5}
$$

which, in the case of a bivariate polynomial is equivalent to

$$
\mu = -\frac{\left[\mathbf{A}_{2,2}\mathbf{b}_1 - \mathbf{A}_{1,2}\mathbf{b}_2, \quad \mathbf{A}_{1,1}\mathbf{b}_2 - \mathbf{A}_{1,2}\mathbf{b}_1\right]^{\mathrm{T}}}{\mathbf{A}_{1,1}\mathbf{A}_{2,2} - \mathbf{A}_{1,2}^2}.
\tag{6}
$$

The location of the minimum of the quadratic μ yields a subpixel displacement vector (because we assumed $(\hat{u}, \hat{v}) = (0, 0)$ to solve the problem, when in reality $(\hat{u}, \hat{v})$ can be at any position). We simply need to add this displacement to the original location of the pixel level minimum. This procedure is quite vulnerable to image noise and flat regions, so we only take that displacement into account if it is less than a pixel away from the original distance minimum location: if $\|\mu\| \leq 1$, we update the estimated motion vector $(\hat{u}, \hat{v})$ to $(\hat{u}, \hat{v}) + \mu$.

Even though this procedure is supposed to help with the upsampling of motion vectors from one level of the pyramid to the next, we have found very few visual improvements to the overall result in our experiments. Subpixel alignment is not applied at the end of the finest pyramid level: because we're dealing with downsampled Bayer to grayscale images, having subpixel motion vectors at the last scale could imply having pixels aligned to pixels from different Bayer CFA channels, which would induce significant color shifts (which we verified in our experiments).

The pseudo-code of the overall alignment strategy of the HDR+ pipeline is described in Algorithm 1.

4 Fourier Tile-based Merging

For a given reference tile, we now have a set of corresponding tiles (one per alternate frame according to the results of the alignment step). If these tiles feature the same image content, i.e. if motion (or lack thereof) is correctly estimated, intensity differences between them can only be attributed to noise, and we can merge these tiles to obtain a single, temporally denoised tile.

This is of course an ideal case, and in practice, tiles can also differ because of errors in the motion estimation. The method presented in Section 3 has several limitations that can lead to alignment errors:

- The alignment is tile-based, so pixels that belong to different objects can be part of the same tile and therefore be attributed the same motion.

- Motion is estimated locally as the most likely translation of a tile of fixed size in the image plane, according to a minimal L1 or L2 norm. This means that any object motion that is not parallel to the image plane is not guaranteed to be correctly approximated.

Algorithm 1: HDR+ tile-based alignment

> **input** : $\{I_0, \ldots, I_{N-1}\}$ burst of N downsampled grayscale images $\qquad I_0$=*reference image*
>
> **input** : $\{p_0, \ldots, p_3\} \in \{1, 2\}^4$ norm power at each pyramid level $\qquad$ *Typically* $\{2, 2, 2, 1\}$
>
> **output:** Sets of aligned tiles at reference image location (i, j) $\{T_0(i, j), \ldots, T_{N-1}(i, j)\}$
>
> **foreach** $k \in \{0, \ldots, N-1\}$ **do**
> > Compute G_k^l, $l \in \{0, \ldots, 3\}$ $\qquad$ *4-level coarse-to-fine Gaussian pyramid of I_k, $G_k^3 = I_k$.*
>
> **foreach** $k \in \{1, \ldots, N-1\}$ **do**
> > $(u_{-1}, v_{-1})(i, j) \leftarrow (0, 0) \ \forall i, j$ $\qquad$ *Initial guess at coarse scale: no offset.*
> >
> > **foreach** *pyramid level* $l \in \{0, \ldots, 3\}$ **do**
> > > Divide G_0^l in equally spaced tiles T_0^l of size $n_l \times n_l$
> > >
> > > **if** $l > 0$ **then**
> > > > Upsample the previous level alignments $(u_{l-1}, v_{l-1})(i, j)$, $\forall i, j$
> > >
> > > **foreach** *reference tile T_0^l at location (i, j)* **do**
> > > > **if** $l > 0$ **then**
> > > > > Update the upsampled $(u_{l-1}, v_{l-1})(i, j)$ by keeping the alignment that minimizes $D_1(u_{l-1}, v_{l-1})$ among those of the 3 nearest coarse-scale tiles
> > > >
> > > > Get all possible (u, v) locations in G_k^l of tiles of size $n_l \times n_l$ in a search area of size $(n_l + 2r_l) \times (n_l + 2r_l)$ centered around $(i + u_{l-1}, j + v_{l-1})$ $\qquad r_l$: *search radius.*
> > > >
> > > > Compute all possible distances $D_{p_l}(u, v)$, $p_l \in \{1, 2\}$ $\qquad$ *Equation (1).*
> > > >
> > > > $(u_l, v_l)(i, j) \leftarrow (u_{l-1}, v_{l-1})(i, j) + \mathrm{argmin}_{u,v}\, D_{p_l}(u, v)$ $\quad$ *Add computed displacement to initial guess.*
> > > >
> > > > **if** $l < 3$ **then**
> > > > > Compute subpixel displacement vector $\mu(u_l, v_l)$
> > > > >
> > > > > **if** $||\mu|| < 1$ **then**
> > > > > > $(u_l, v_l)(i, j) \leftarrow (u_l, v_l)(i, j) + \mu$
> >
> > Associate to the reference tile at finest scale T_0^3 at location (i, j) the tile T_k^3 of I_k at location $(i + u_3, j + v_3)$

- Depending on the selected parameters (successive downsampling factors, tile sizes, search radii and norm types) versus the actual motion, motion estimation could be stuck in a local mininum. That said, since most real world scenes will feature motion that is compatible with the short exposure time of the burst, a good choice of parameters will mitigate that problem.

- The motion estimation is sensitive to noise: the higher the noise level with respect to the image content, the higher the likelihood of having an incorrect tile minimizing the distance.

- As with many motion estimation techniques, it does not bode well with occlusions: areas that are not occluded in the reference frame but are in the alternate frame will probably yield incorrect motion vectors.

Such errors are expected given the relatively simple nature of the alignment method. Keep in mind that it is designed to run quickly on systems like smartphone systems on a chip (SoCs). Moreover, this fast, imperfect motion estimation can be sufficient for subsequent processing steps, provided the merging step takes those potential alignment errors into account. In this section, we will discuss the strategy employed by the authors of [9] to perform temporal denoising while being robust to alignment errors.

4.1   Noise Level Estimation

For a set of tiles, the first prerequisite of the merging step is an estimation of the noise level. Since the noise profile is considered identical for all the images of the burst, we perform that estimation on the reference image.

In the HDR+ pipeline, raw image noise follows the Poissonian-Gaussian model described in [7], where it is composed of two mutually independent parts: a Poissonian signal-dependent component and a Gaussian signal-independent component. Given the properties of Poissonian and Gaussian distributions, the noise variance σ^2 can simply be expressed as an affine function of the signal level x:

$$\sigma^2(x) = \lambda_s x + \lambda_r \tag{7}$$

where the parameter λ_s can be tied to shot noise, and λ_r can be tied to read noise [17].

There are multiple ways to obtain these two noise curve parameters:

- One can take one or several pictures of a scene that features multiple uniform (typically gray) areas and perform mean and variance measurements of intensities [7, 10]. λ_s and λ_r can then be computed from simple linear regression.

- In [9], the (λ_s, λ_r) tuple is claimed to only depend on the analog and digital gain settings selected at capture time. Since the authors implemented their pipeline on a specific set of smartphones and sensors, they could likely use a look-up-table that gives them the appropriate per-camera (λ_s, λ_r) tuple as a function of the applied gain settings. In some raw DNG files, λ_s and λ_r can be directly found in the `NoiseProfile` DNG tag[8].

- The `NoiseProfile` tag is not present in all DNG files, and separate analog and digital gain values are not directly accessible in image metadata. A single value is usually specified: ISO, which combines analog and digital gain differently from one manufacturer to another. This makes it difficult to extract noise curve parameters *a posteriori*, especially given that different sensors can have different shot versus read noise curves [17]. That said, even though they might not correspond to the actual values of λ_s and λ_r of the image, we found that computing a (λ_s, λ_r) tuple from the image ISO and baseline values of λ_s and λ_r for an image at ISO=100 produces equally pleasing results for the rest of the merging step. We use the following formula:

$$
\begin{aligned}
\sigma^2\left(\alpha_{\text{ISO}}x\right) &= \left(\alpha_{\text{ISO}}\right)^2 \sigma^2(x) \\
&= \left(\alpha_{\text{ISO}}\right)^2 \left(\lambda_{s_{\text{ISO}_{100}}} x + \lambda_{r_{\text{ISO}_{100}}}\right) \\
&= \alpha_{\text{ISO}}\lambda_{s_{\text{ISO}_{100}}} \alpha_{\text{ISO}}x + \left(\alpha_{\text{ISO}}\right)^2 \lambda_{r_{\text{ISO}_{100}}} = \lambda_s^{'} x^{'} + \lambda_r^{'},
\end{aligned}
\tag{8}
$$

where $\alpha_{\text{ISO}} = \frac{\text{ISO}}{\text{ISO}_{100}}$ is the ratio between the ISO setting used at capture time and ISO_{100} (considering that $\text{ISO}_{100} = 100$ is the baseline ISO where no gain is applied), $x^{'} = \alpha_{\text{ISO}}x$ is the image we're actually observing, and $(\lambda_{s_{\text{ISO}_{100}}}, \lambda_{r_{\text{ISO}_{100}}})$ are the baseline noise curve parameters at ISO_{100} (these could be known for a given camera, averaged from a certain amount of ISO-normalized images, or arbitrary).

- Another solution is to use an off-the-shelf noise curve estimation algorithm directly on the input image. These algorithms usually revolve around finding homogeneous regions within the image and computing the variance and means of said regions [4, 18]. Using a noise curve estimation algorithm would effectively allow the HDR+ burst denoising algorithm to run blind,

---

[8]https://wwwimages2.adobe.com/content/dam/acom/en/products/photoshop/pdfs/dng_spec_1.4.0.0.pdf

as all necessary information is stored in the raw Bayer data. That said, extra care should be taken to verify the robustness of these algorithms and the consistency of their outputs. In our experiments, we found that sticking to image metadata or computing (λ_s, λ_r) from ISO and baseline values produced more consistent results, all while requiring less computational time and fewer external dependencies.

In order to avoid potential problems due to the undersampled nature of the red and blue channels of a Bayer pattern, the noise estimation and the remainder of the merging method are performed separately for each color plane (the two green channels are also treated independently). Since we computed pixel-level alignment for tiles of downsampled grayscale images, we can actually assign the same tile size and motion vectors to each individual color plane.

For computational efficiency, even though the noise is claimed to follow a signal-dependent model across the reference image, the noise variance is actually considered signal-independent within each reference image tile. This means that for a given tile T, a single intensity value ρ is used to evaluate the noise model. Instead of taking the average of all intensities within the tile, the authors of [9] decided to consider the root-mean-square (RMS) of the tile

$$\rho(T) = \text{RMS}(T) = \left(\frac{1}{n^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} T(i,j)^2 \right)^{\frac{1}{2}}. \tag{9}$$

The reasoning being that given two tiles of the same average, one having a higher contrast than the other, the higher contrast tile will have a higher root-mean-square (and thus a higher estimated noise variance), which will allow a more aggressive temporal denoising.

4.2 Pairwise Wiener Temporal Denoising

Given a set of tiles (the reference tile and the alternate tiles selected by motion estimation), each alternate tile T_z is compared to the reference T_0. The difference between the two tiles $D = T_0 - T_z$ is computed, and that difference is compared to the estimated noise variance $\sigma^2(\rho(T_0))$. That comparison is then used to compute a weighted average of T_z and T_0 (if the difference between the two tiles is far greater than the estimated noise level, the alternate tile will have a very small weight and we fall back to the reference).

The final temporally denoised result is the average of all pairwise merges. This method is actually performed in the frequency space (i.e. via the 2D DFT of tiles), which allows each spatial frequency bin to be treated individually. This can be useful in cases where alignment failure is partial (i.e. some frequencies match well while others don't) as not all the content of the tiles will be discarded.

Mathematically,

$$\tilde{T}_0(\omega) = \frac{1}{N} \sum_{z=0}^{N-1} (1 - A_z) T_z(\omega) + A_z(\omega) T_0(\omega), \tag{10}$$

where the shrinkage operator $A_z(\omega)$ is similar to a Wiener filter

$$A_z(\omega) = \frac{|D_z(\omega)|^2}{|D_z(\omega)|^2 + c\sigma^2(\rho(T_0))}, D_z(\omega) = T_0(w) - T_z(\omega), \tag{11}$$

where c is defined in the original article as a scaling and tuning factor. For the sake of simplicity in this publication, we will set $c = k\tau$ where k is the scaling factor and τ is the tuning factor that effectively controls the strength of temporal denoising. In [9], k is fixed to $n^2 \times 1/4^2 \times 2$, where n is the tile length. However, the justification for this in the article is questionable, because they are

trying to scale a single variance value computed in the spatial domain to a per-frequency-bin Wiener filter of the squared difference of two images. Still, we leave the value of k as is, since its influence can be superseded by simply changing the values of τ. The pseudo-code of temporal denoising is described in Algorithm 2.

Remark. *This 2D frequency-based denoising technique bears similarities with the burst deblurring method of Fourier Burst Accumulation [6, 1]. However, the comparison of 2D frequency bins in HDR+ serves a different purpose: if T_0 is sharp, and it is still matched to a blurry tile T_z after alignment, the frequency content corresponding to the blurry elements ot T_z will be discarded, but only because it does not match that of T_0. On the contrary, if we have a blurry T_0, but a sharp T_z, there is also a strong frequency mismatch and we will not merge that content either. In effect, the Wiener filter does not introduce additional blur from other images, but it does not remove blur either.*

If $\tau \to 0$, $\tilde{T}_0(\omega) \to T_0(\omega)$: we stick to the reference frame and the result is not temporally denoised. If $\tau \to +\infty$, $\tilde{T}_0(\omega) \to \frac{1}{N}\sum_{z=0}^{N-1} T_z(\omega)$: the result is equivalent to an average of all aligned frames in the spatial domain, which can showcase alignment errors. These edges cases can be observed in Figure 4.
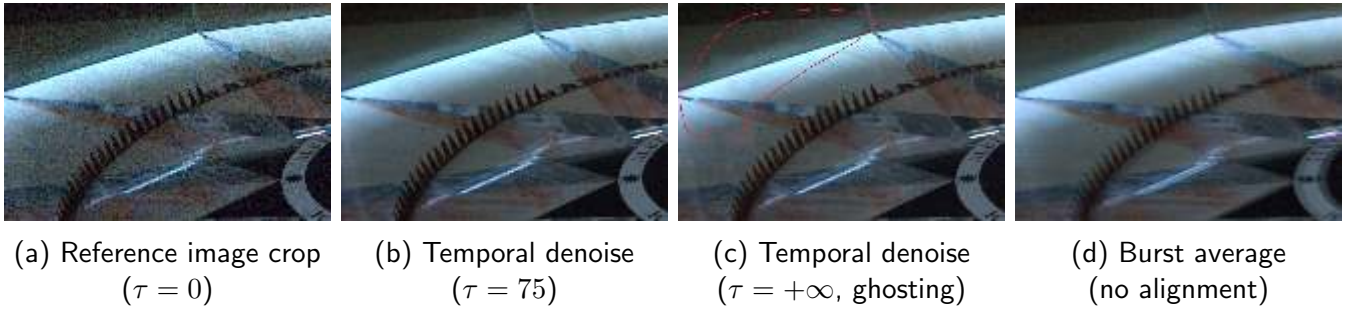

| (a) Reference image crop ($\tau = 0$) | (b) Temporal denoise ($\tau = 75$) | (c) Temporal denoise ($\tau = +\infty$, ghosting) | (d) Burst average (no alignment) |

Figure 4: Impact of the tuning factor τ on the result of temporal denoising.

4.3 Wiener Spatial Denoising

In addition to temporal filtering, spatial filtering is performed on the 2D DFT of tiles in order to remove some of the residual noise. Since we expect the noise level to be significantly reduced after temporal denoising, the estimated noise variance is updated to $\sigma^2(\rho(T_0))/N$ (this implies a perfect averaging of all N frames, which is not the case in practice but allows for a more conservative spatial denoising). We compute a shrinkage operator similar to the one used for temporal denoising and apply it to the spatial frequency coefficients

$$\hat{T}_0(\omega) = \frac{\left|\tilde{T}_0(\omega)\right|^2}{\left|\tilde{T}_0(\omega)\right|^2 + f(\omega)\frac{\sigma^2(\rho(T_0))}{N}}\tilde{T}_0(\omega), \tag{12}$$

where $f(\omega)$ is a noise shaping function; in [9], it is a piecewise linear function that increases the "effective noise level" for higher spatial frequencies, tuned to subjectively maximize image quality. As we do not know the specifics of said shaping function, we simply replaced it with $f(\omega) = \gamma |\omega|$, which also effectively increases the noise level for higher frequencies, and features a scaling and tuning factor γ. We set $\gamma = \frac{k}{2}s$, where k is the scaling factor defined in Section 4.2 and s controls spatial denoising strength.

Algorithm 2: HDR+ per-channel, tile-based, pairwise temporal denoising

input : Set of aligned tiles $\{T_0, \ldots, T_{N-1}\}$ of size $n \times n$ at reference image location (i,j)
  (all tiles correspond to a single channel of the Bayer image)

input : Noise curve parameters λ_s, λ_r, temporal denoising tuning factor c

output: Single denoised tile $\tilde{T}_0$ at reference image location (i,j)

$\rho \leftarrow \text{RMS}(T_0) = \left(\frac{1}{n^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} T_0(i,j)^2 \right)^{\frac{1}{2}}$ *Compute the root-mean-square of T_0.*

$\sigma^2 \leftarrow \lambda_s \rho + \lambda_r$ *Compute the noise variance assigned to T_0.*

foreach $z \in \{0, \ldots, N-1\}$ **do**
  $\quad T_z(\omega) \leftarrow \text{FFT}(T_z)(\omega)$ *DFT of $T_z(x,y)$.*

$\tilde{T}_0(\omega) \leftarrow T_0(\omega) \; \forall \omega$

foreach $z \in \{1, \ldots, N-1\}$ **do**
  foreach *frequency bin* $\omega \in \{0, \ldots, n-1\}^2$ **do**
    $\quad D_z(\omega) \leftarrow T_0(\omega) - T_z(\omega)$
    $\quad A_z(\omega) \leftarrow \frac{|D_z(\omega)|^2}{|D_z(\omega)|^2 + c\sigma^2}$
    $\quad \tilde{T}_0(\omega) \leftarrow \tilde{T}_0(\omega) + (1 - A_z)T_z(\omega) + A_z(\omega)T_0(\omega)$ *Sum pairwise temporal denoisings.*

$\tilde{T}_0(\omega) \leftarrow \frac{1}{N}\tilde{T}_0(\omega)$ *Average pairwise temporal denoising.*

$\tilde{T}_0(i,j) \leftarrow \text{FFT}^{-1}(\tilde{T}_0)(i,j)$ *Inverse DFT of $\tilde{T}_0(\omega)$.*

Mathematically, if $s = 0$ we do not perform any spatial denoising. The higher the s, the more aggressively we filter higher spatial frequencies and obtain a lower-frequency final image (with little to no edges and textures within a tile in extreme cases). We typically set it to 0.1 to remove a bit of high frequency residual noise without loosing too much detail. The influence of s on the spatial denoising can be observed in Figure 5.

(a) Reference image   (b) Temporal denoise $(s = 0)$ (c) Temporal + spatial denoise $(s = 10)$ (d) Temporal + spatial denoise $(s = 10^{12})$
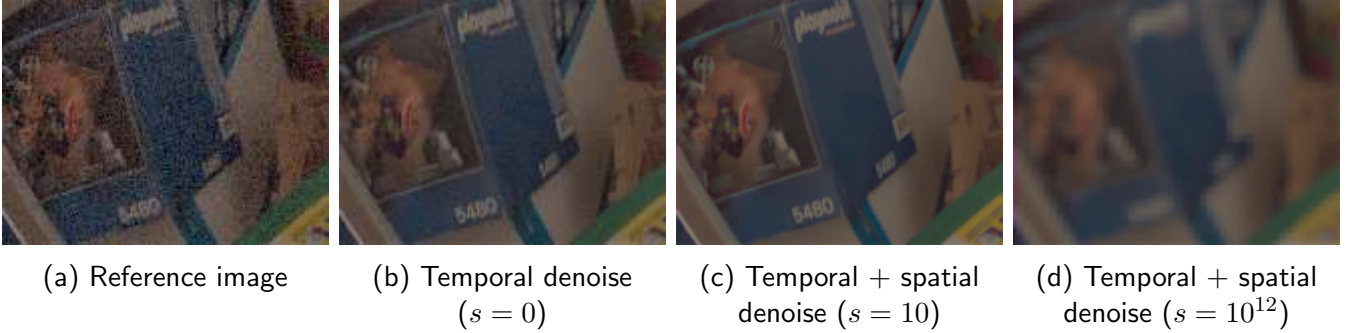
Figure 5: Impact of the tuning factor s on the result of spatial denoising

This frequency-based denoising technique (and particularly our simplified implementation) is not expected to be as efficient as state-of-the-art spatial denoising algorithms [5, 14, 19]. However, this step is used as a very computationally light complement to temporal denoising (since we are already in frequency space), which can be tuned to be fairly gentle and not remove image detail.

4.4 Overlapped Tiles and Raised Cosine Window

The steps described so far are the following: we start by dividing the reference image in equally spaced tiles; we then associate each reference tile to a specific stack of alternate tiles (one for each

other image of the burst according to the result of the alignment step); after that, each stack is merged using the strategy described in the previous steps of Section 4. Once all stacks are merged, we obtain a set of denoised tiles of the same number and disposition we had after dividing the reference frame, which we can call tiles of the *merged* image. Unfortunately, these steps are not sufficient to create both a less noisy image and a realistic looking one. Because we merge each stack independently of others, and because different stacks have different image content, a different noise variance estimation, and can feature different alignment errors, continuity between adjacent tiles is not guaranteed. Additional artifacts can be seen on tile edges, because the merging operation is performed in the DFT domain (we compute the FFT of a tile stack, merge it, and then compute the inverse FFT to obtain a merged tile). These issues can be seen in Figure 6.

(a) Noisy reference image crop
(b) Merged image crop. Discontinuities between tiles and artifacts on tile edges due to DFT can be observed
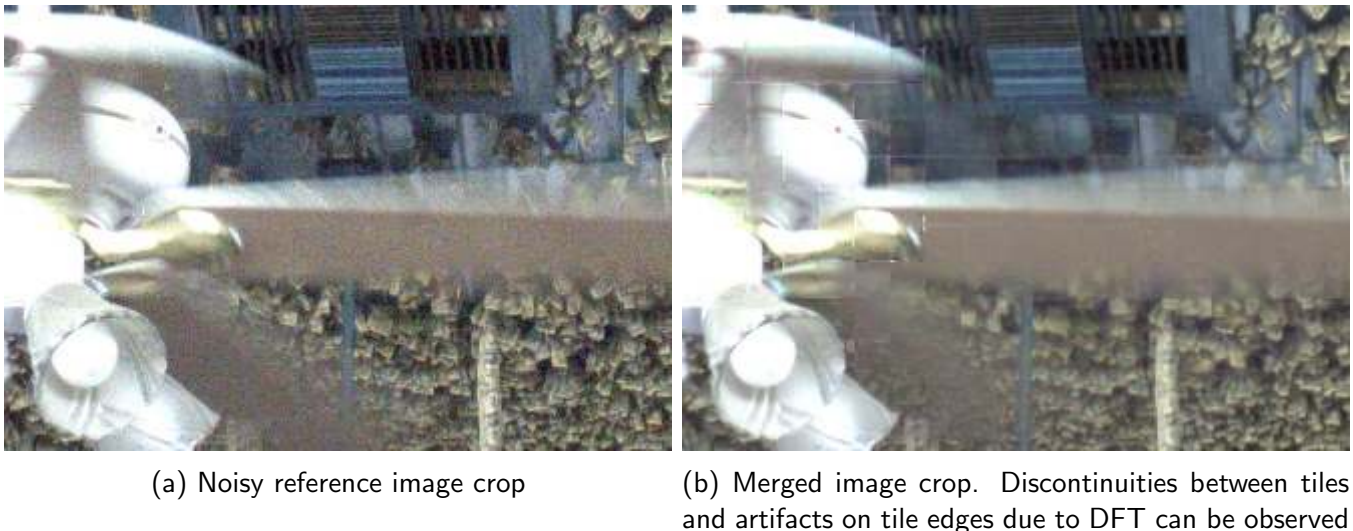
Figure 6: Results obtained with the steps described so far

In order to create smoother images with less visually noticeable artifacts while retaining image detail, the solution of the authors of [9] is twofold:

- **Tiles** (for both the alignment and merging steps) are **overlapped by half in each spatial dimension**. This means that the total number of tiles (and thus the number of operations for alignment and merging) is actually multiplied by a factor of 4.

- The window used for blending is a **modified raised cosine window**, defined in 1D as

$$w(x) = \frac{1}{2} - \frac{1}{2}\cos\left(2\pi\left(x + \frac{1}{2}\right)/n\right), 0 \leq x \leq n - 1. \tag{13}$$

If x is a vector of positions from 0 to $n - 1$, this window function is centered, and has nonzero values at 0 and $n - 1$ which means that content on edges is not fully discarded. As can be seen in Figure 7a, another interesting property is that when they overlap by half, the sum of two such windows is always equal to 1. When dealing with tiles that overlap by half in two spatial dimensions, the actual window used for blending (visible in Figure 7b) is the product of the 1D version in each dimension

$$w(i,j) = \left(\frac{1}{2} - \frac{1}{2}\cos\left(2\pi\left(i + \frac{1}{2}\right)/n\right)\right)\left(\frac{1}{2} - \frac{1}{2}\cos\left(2\pi\left(j + \frac{1}{2}\right)/n\right)\right), 0 \leq i, j \leq n - 1. \tag{14}$$

This window allows the smooth blending of the overlapped tiles and has the added benefit of removing DFT artifacts on tile edges.

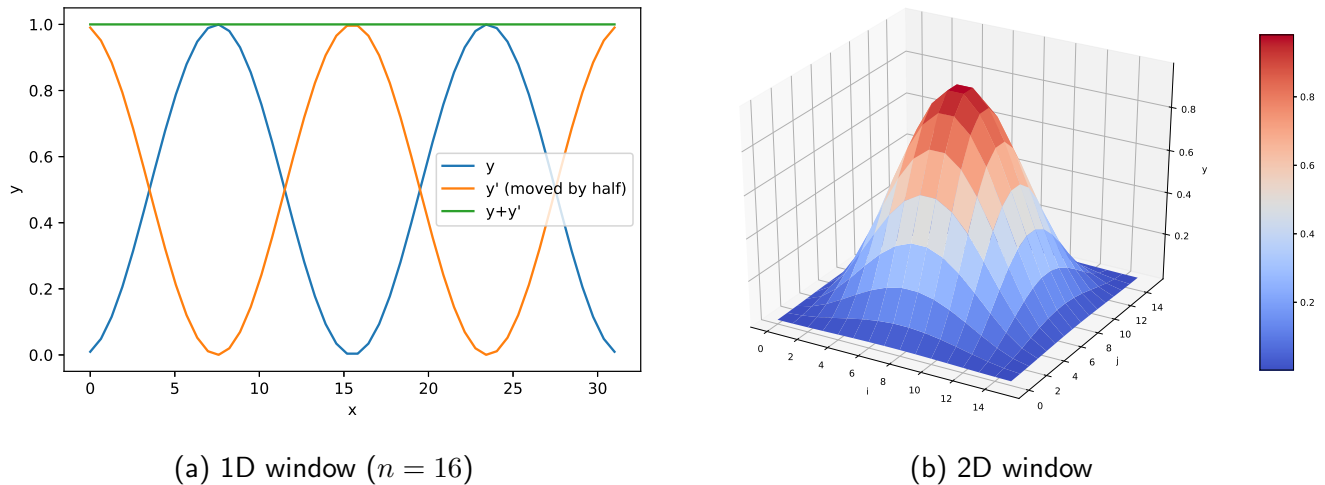(a) 1D window ($n = 16$)   (b) 2D window

Figure 7: Modified raised cosine window.

Going back to our real burst denoising example, Figure 8 showcases that using the aforementioned blending technique does create a more pleasing image without loosing image detail.
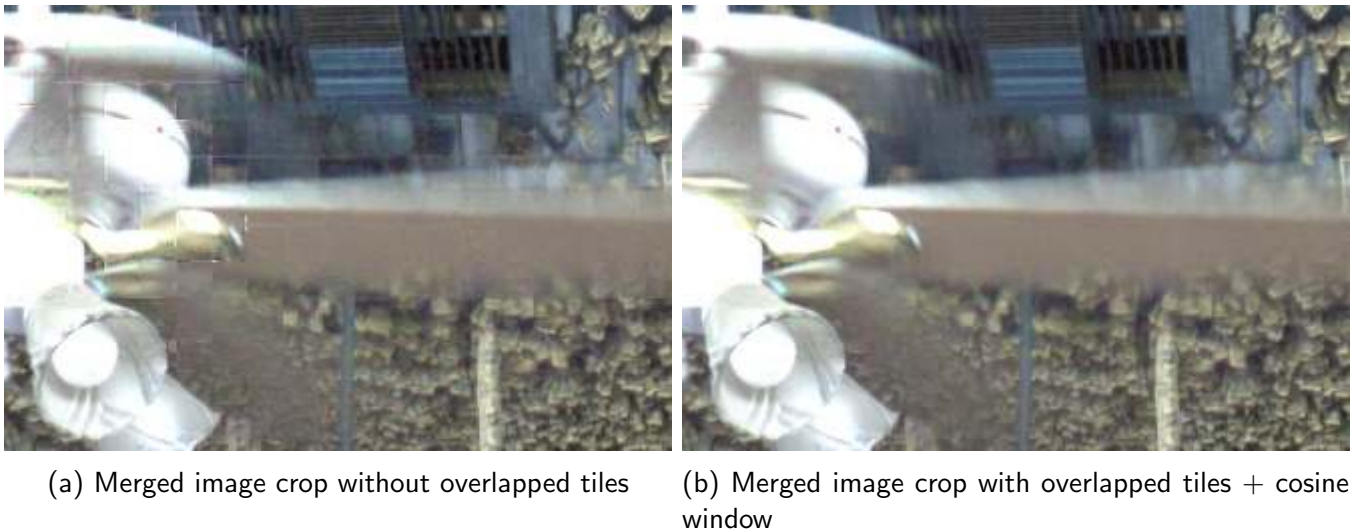

(a) Merged image crop without overlapped tiles   (b) Merged image crop with overlapped tiles + cosine window

Figure 8: Using overlapped tiles and the modified raised cosine window prevents observable discontinuities and edge artifacts while retaining image detail.

5 Finishing

After the alignment and merging steps, we obtain a single raw image based on the reference but with significantly less noise. The system described in [9] being a full camera pipeline, the subsequent steps that lead to a final `.jpg` image are also presented. Even though this article mainly focuses on the temporal denoising part of the HDR+ solution, this section discusses the remainder of the image processing pipeline.

5.1   Google's Pipeline

The system described in [9] being a full camera pipeline, the subsequent steps that lead to the final image are also presented:

- Black level subtraction: saved values of additional sensor pixels that are shielded from the light are subtracted so that image pixels that do not receive any light have a 0 value.

- Lens shading correction: areas on the image corners that are darker are brightened. Low-resolution lens shading maps that are provided by the ISP are used for that compensation.

- White balance: colors are shifted so that objects human observers would perceive as gray actually appear as gray in the final image. Gains are applied to each one of the 4 RGGB channels individually, and those gains are provided by the ISP.

- Demosaicking: the raw Bayer image where each pixel corresponds to a single color channel is converted to an RGB image of the same size where each pixel has 3 color channels. The raw image being undersampled, an interpolation algorithm is used (the algorithm in the HDR+ pipeline is said to use a combination of techniques featured in [8] and is most likely proprietary).

- Chroma denoising: even after the temporal and spatial denoising steps, the image can still contain artifacts such as red and green blocks in dark areas, especially in the case of low-light images. To that end, the HDR+ authors apply a sparse 3×3 tap non-linear kernel in two passes to the image in the YUV space, acting as an approximate bilateral filter.

- Color correction: a 3×3 matrix supplied by the ISP is used to convert the image from sensor RGB to linear sRGB values.

- Dynamic range compression / local tone mapping: aside from burst denoising, the HDR+ pipeline aims at improving the dynamic range of the final image in specific scenarios (hence its name): in scenes that feature a high dynamic range, darker areas of the image must be brightened, while brighter content must remain unsaturated and local contrast must be preserved. To that end, a local tone mapping method derived from the exposure fusion algorithm [16, 11] is employed. However, since this method requires several images of different brightness (usually several captures at different exposure times), two gamma-corrected "synthetic exposures" are created from the intermediate result: one short (a gamma-corrected grayscale version of the current image, which features the exposure time used during the burst capture) and one long exposure (a gamma-corrected grayscale version of the current image with a gain applied according to the auto-exposure algorithm specific to that pipeline). From these two images, the exposure fusion algorithm uses image pyramids to create a smoothly blended image where each pixel is best-exposed. The gamma correction is then inverted and the image re-colorized by keeping the chroma ratios of the intermediate image.

- Dehazing: an extra global tone curve that pushes low pixel values even lower while preserving highlights and midtones is applied in order to mitigate the effect of veiling glare.

- Global tone adjustment / gamma curve: an S-shaped contrast enhancing tone curve is concatenated to the sRGB color component transfer function to transform the linear sRGB image to a nonlinear sRGB image where contrast is increased.

- Chromatic aberration correction: longitudinal and lateral chromatic aberration are corrected by replacing chroma values on high contrast edges by those of nearby pixels that are less likely to be affected by chromatic aberration.

- Sharpening: the authors implement unsharp masking[9] through a sum of Gaussian kernels constructed from a 3-level convolution pyramid.

- Hue-specific color adjustments: some custom transformations are applied to the image colors (shifting bluish cyans and purples towards light blue, and increasing the saturation of blues and greens generally), which is said to make vegetation and blue skies more visually appealing.

- Dithering: to mitigate quantization artifacts when reducing from 10/12 to 8 bits per pixel (sRGB images are typically destined for viewing on 8-bit displays), dithering is implemented by adding blue noise from a precomputed table.

- JPEG quantization and compression: the image is quantized and encoded to 8 bits where the lossy compression is set to a quality level of 95, resulting in the final `.jpg` file.

All these steps are performed via software (i.e. in a smartphone, these operations can run on the CPU/GPU and not necessarily on the ISP).

5.2   Our Simplified Pipeline

Since we decided to mainly focus on the denoising part of the HDR+ algorithm for this article, we implemented a simpler finishing pipeline. It still features parts of Google's own solution, and it is presented as a proof of concept, to showcase some of the decisions they made when designing their pipeline. It does not aim for parity with Google in terms of visual quality, as it would require a considerable amount of additional development time and a lot of trial and error, especially given the relatively sparse description of these steps in the original article.

Using rawpy, a Python wrapper of the LibRaw library[10] we transform a temporally and spatially denoised 16 bit Bayer array (even though smartphone raw images typically have a 10 or 12 bit precision, they are often stored on 16 bit arrays) into a 16 bit linear RGB image of the same resolution with the following operations:

- **Black level subtraction**

- **White balance**: rawpy uses the gains stored in the `As Shot Neutral` Exif metadata tag in the `.dng` file of the reference image to apply scales to the red and blue Bayer channels.

- **Demosaicking**: the raw Bayer image where each pixel corresponds to a single color channel is converted to a RGB image of the same size where each pixel has 3 color channels. We use the AHD algorithm [13] since it was both used for burst fusion comparisons in the HDR+ supplement (where it was deemed "representative of the algorithms used by mobile ISPs") and available in rawpy. That algorithm is not the proprietary one used in Google's implementation.

- **Color correction**: rawpy uses image metadata such as the `Color Matrix` Exif tag in the `.dng` file of the reference image to convert the image from sensor linear RGB to standard linear sRGB color space.

Once we're done with rawpy postprocessing, we apply additional custom operations:

- **Hdr tone mapping**: we use a technique close to the one employed in [9], where two synthetic exposures are created from a single image and combined with exposure fusion [16] (a similar strategy is extended and demonstrated in [12]). From the result of all previous steps (which

---

[9]https://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/unsharpmask/index.html
[10]rawpy: RAW image processing for Python, a wrapper for LibRaw, https://pypi.org/project/rawpy/, 2014

stems from an underexposed burst), we get a grayscale image by averaging the 3 color channels and then synthesize two exposures: we apply the standard sRGB gamma correction to get a short exposure, and we apply a gain followed by the same gamma correction to synthesize the long exposure. We then perform exposure fusion using the OpenCV `mergeMertens` implementation[11]. Given that we are blending grayscale images, and that in their implementation, the authors of [9] use "a fixed weighting function of luma that favors moderately bright pixels", we found that only taking the "well-exposedness" weights of exposure fusion into account for blending (we set the `exposure_weight` parameter to 1 and `contrast_weight` and `saturation_weight` to 0 when calling the `createMergeMertens` function) produced convincing results (Figure 9 shows an example of the applied exposure fusion). We undo the gamma

(a) Synthetic short exposure   (b) Long exposure (gain=15)   (c) Exposure fusion result
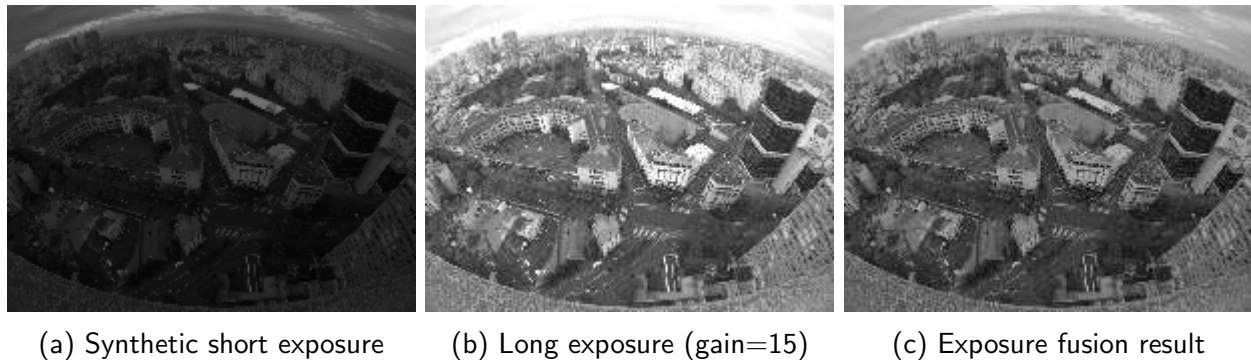
Figure 9: Dynamic range compression / local tone mapping.

correction of the fused grayscale image, and compute a per-pixel scaling with the element-wise division of the result by the original grayscale image. We then apply that scaling to each channel of the underexposed RGB image to obtain the tone-mapped result.

- **Contrast enhancement / global tone mapping**: our next step is to increase image contrast by applying the following simple S-shaped function to each color channel

$$y = \max(0, \min(x - \alpha \sin(2\pi x), 1)).$$ (15)

This makes dark areas of the image darker and bright areas brighter, thus enhancing contrast. As illustrated in Figure 10, the α parameter must be carefully selected to avoid completely black or white regions while retaining a visually pleasing result. Although it is probably different from the one used in Google's own HDR+ implementation, this S-shaped contrast enhancing curve is actually very similar to the one referenced and used in another Google publication [2].

- **Gamma compression**: pixel values are shifted from linear to non-linear sRGB space in order to both reduce the required bandwidth and produce a more pleasing and identifiable look, taking advantage of the human visual sensitivity which is also non-linear (for example, humans are more sensitive to brightness differences in darker tones versus brightness differences in brighter tones). The sRGB transfer function is defined by the IEC 61966-2-1 international standard[12] as

$$V_{out} = \begin{cases} 12.94 V_{in}, & 0 \le V_{in} \le 0.0031308, \\ 1.055 V_{in}^{\frac{1}{2.4}} - 0.55, & 0.0031308 < V_{in} \le 1. \end{cases}$$ (16)

---

[11]https://docs.opencv.org/master/d7/dd6/classcv_1_1MergeMertens.html
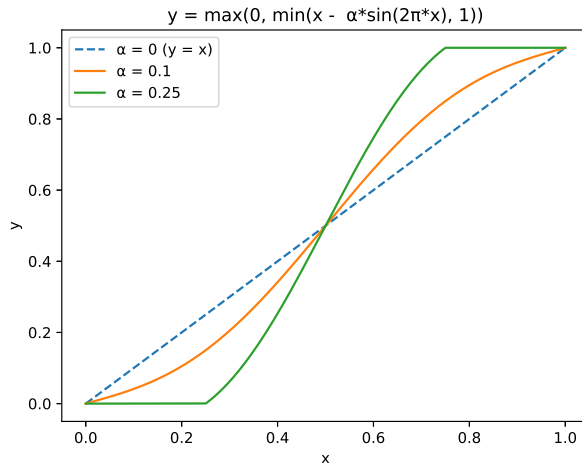[12]http://www.color.org/chardata/rgb/srgb.xalter

Figure 10: S-shaped contrast enhancement curve.

- **Sharpening**: we get a final sharper image by computing the average of 3 images obtained via unsharp masking

$$I_{sharp} = \frac{1}{3} \sum_{m=1}^{3} I_{sharp_m}, \tag{17}$$

where

$$I_{sharp_m}(x,y) = \begin{cases} I(x,y) + \alpha_m(I(x,y) - I_{blur}^{\sigma_m}(x,y)) & \text{If } |I(x,y) - I_{blur}^{\sigma_m}(x,y)| > \tau_m, \\ I(x,y) & \text{otherwise,} \end{cases} \tag{18}$$

where $I_{blur}^{\sigma_m}$ is the result of the convolution of I with a Gaussian kernel of standard deviation σ_m, α_m controls the sharpening strength and τ_m is a threshold that controls which pixels will actually be sharpened. We empirically found that setting $\alpha_m \in \{1, 0.5, 0.5\}$, $\sigma_m \in \{1, 2, 4\}$ and $\tau_m \in \{2\%, 4\%, 6\%\}$, $m \in \{1, 2, 3\}$ produced visually pleasing results (image that looks a bit sharper while remaining natural-looking, little to no noise reintroduced after sharpening homogeneous regions). This mask is certainly different from the one employed in [9] given that its parameters are unknown.

- **JPEG quantization and compression**: the image is quantized and encoded to 8 bits, resulting in the final `.jpg` file. We set the quality level of the lossy compression to 100 to minimize compression artifacts.

Although these steps produce less visually pleasing images than the full HDR+ finishing pipeline, they still showcase one of its most defining aspects: HDR tone mapping from a single underexposed merged image. A full or partial combination of these steps can be used for any raw .dng image, including the noisy reference image, or intermediate results of Google's own implementation. This will come in handy for visual comparisons, which we'll discuss in Sections 6.2.2 and 6.3.

6 Comparison to the Original HDR+ Implementation

6.1 The HDR+ Dataset

In 2018, [9] was completed by a very large dataset captured with several smartphones, simply named the HDR+ dataset[13]. It is comprised of 3640 raw bursts (28461 images total), for a total of 765 GB. Each burst contains the following files:

- a set of raw images stored in the open and commonly used Adobe Digital Negative (`.dng`) format, which can be read by rawpy and contains useful metadata such as exposure time, ISO, black level as well as white balance gains estimated by the camera.

- additional metadata stored in separate files, such as maps for lens shading correction and color correction matrices.

A curated subset of 153 bursts is also available. Since the HDR+ algorithm is not learning-based, we mostly focused on that subset for the purposes of this article.

Two sets of Google's own results are available on the dataset website: the 2016 results, said to correspond to the algorithm and parameters described in [9], while the 2017 ones correspond to a more recent version of the pipeline with "algorithm refinements and updated tuning". For each raw burst, a set of results comes with the following data:

- A raw `merged.dng` file which is said to correspond to the result of the alignment and temporal denoising steps (the dataset authors claim no additional processing of the raw image).

- A `final.jpg` file, which is obtained after the full finishing pipeline described in Section 5.1 is applied to the merged result.

- A `reference.txt` file that contains the index of the picked reference frame.

6.2 Comparison to Google's Merged Results

The inclusion of Google's raw alignment and temporal denoising results along with the chosen reference frame allows for an apples-to-apples comparison with our open-source Python implementation of the burst alignment and merging algorithm: we can measure similarity between the raw images and/or put the alignment and merging results through the same finishing pipeline for visual comparison.

6.2.1 Statistical Comparison of Raw Merged Images

We first wanted to compare our raw merged results to the 2016 Google ones, because they are said to be the closest to the implementation presented in the article. However, by putting these files through a simplified finishing pipeline (black level subtraction, white balance, demosaicking, color correction, gamma curve), we actually found that it is very likely that the 2016 version of the `merged.dng` files also features spatial denoising, while the 2017 might not. Significant differences can be observed in Figure 11.

On the subject of spatial denoising, the article provides no indication of the slopes of the "piecewise linear function" of spatial frequency (or of any potential tuning of the spatial denoising strength). Therefore, it would be ill-advised to compare the 2016 results to our own. We instead decided to perform comparisons against the 2017 results. Even though they are said to feature different tunings

---

[13]HDR+ Burst Photography Dataset, http://www.hdrplusdata.org/

(a) 2016 "Merged" image (post-processed)   (b) 2017 "Merged" image (post-processed)

Figure 11: The Google 2016 results seem to feature less residual noise except around strong high contrast features / specular highlights, which is typical of spatial denoising as stated in [9].

and algorithm refinements (we do not know all the 2016 tuning factors either), the alignment and merging results seem more in line with what we were able to obtain with our own implementation.

Using the subset of 153 bursts provided in the HDR+ dataset, we thus took the bursts where the raw reference image and the 2017 Google "`merged.dng`" result were of the same size (for an unknown reason, some Google images are larger than the reference), and set out to compute two PSNRs:

- the PSNR between Google's result and the noisier reference image it is based on.

- the PSNR between Google's result and our own result, tuned to match Google's result as much as possible:

  - Coarse-to-fine upsampling factors of the 4-level Gaussian pyramid: 4, 4, 2 (as suggested in the HDR+ supplement).
  - Coarse-to-fine tile sizes: 8, 16, 16, 16 (as suggested in the HDR+ supplement).
  - Type of norm used for tile distance computation: L2 at all levels except the finest, L1 for the finest (as suggested in the HDR+ supplement).
  - Temporal denoising factor τ: 75 (it is claimed to be set to 8 in [9], but we were not able to reproduce similar temporal denoising performance with our own way of computing the noise parameters (λ_s, λ_r) and thus σ^2).

On that topic, Google decided to increase the bit depth (and in turn change the black and white levels) of their raw merged result to put emphasis on the accuracy gained after alignment and temporal denoising, so we had to normalize the images before computing those PSNRs (our merged result has the same black and white levels as the reference image). With these settings, we achieve an approximate +7dB PSNR improvement when compared to the reference on average, our results having higher PSNR for every burst. This suggests that our implementation does perform alignment and temporal denoising somewhat similarly to Google's implementation. Bear in mind that we're trying to be close to images whose tuning and algorithm refinements are unknown. These similarity measurements can be seen in Figure 12.
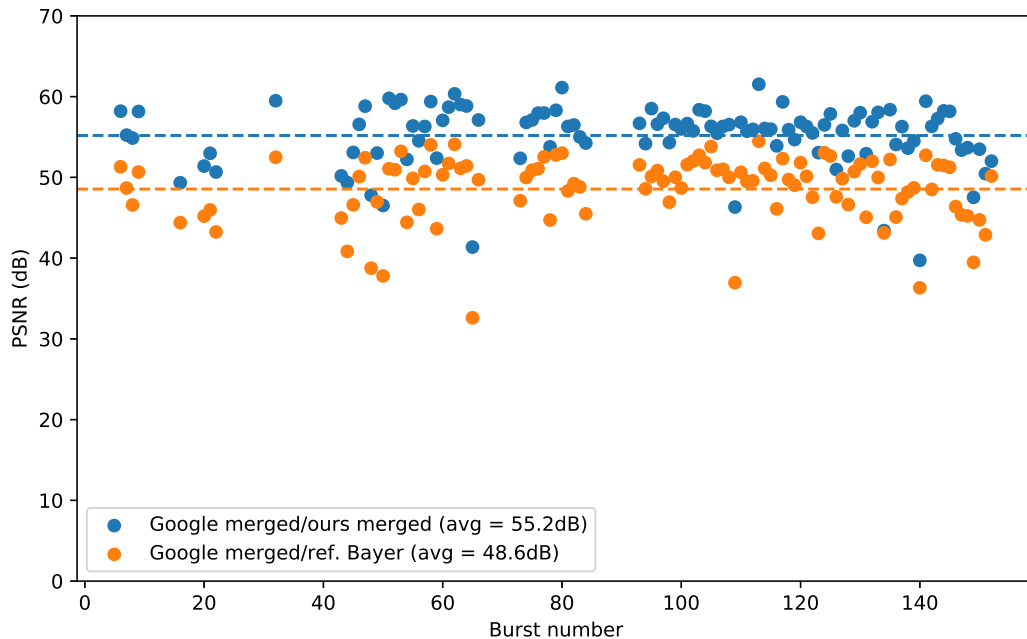
Figure 12: PSNR between Google's and our merged result on the applicable bursts of the "20171106_subset".

6.2.2 Visual Comparison of Minimally Processed Merged Images

If we put the three raw images (the reference image, our merged result, and Google's) through the same minimal finishing pipeline (identical demosaicking + white balance + color matrix + gamma curve), we can visually assess the similarity between Google's implementation of the alignment and temporal denoising steps and our own.

Looking at the subset of 153 bursts, some images seem to have less residual noise on our end, while others look more noisy than Google's result. The most likely candidates for noticeable differences between the two are:

- The way we compute the (λ_s, λ_r) noise curve parameters: we either extract them from `.dng` metadata when possible, or compute them as a function of baseline ISO 100 values and actual ISO (as explained in Equation (8) of Section 4.1), while Google can derive these parameters from the analog and digital gain set at capture time.

- In [9], the authors claim to typically use 16×16 tiles for alignment (at the finest pyramid level) and merging, except for very dark scenes where they use 32×32 tiles. We do not make that distinction and always use 16×16 tiles.

- Google could use different tuning factors from ours (this interacts with the two previous potential differences anyways).

- The so called "algorithm refinements" of the 2017 results we're unaware of.

Selected crops can be observed in Figure 13. Full-size results for the whole subset of 153 bursts can be consulted and downloaded from a Google Photos album[14].
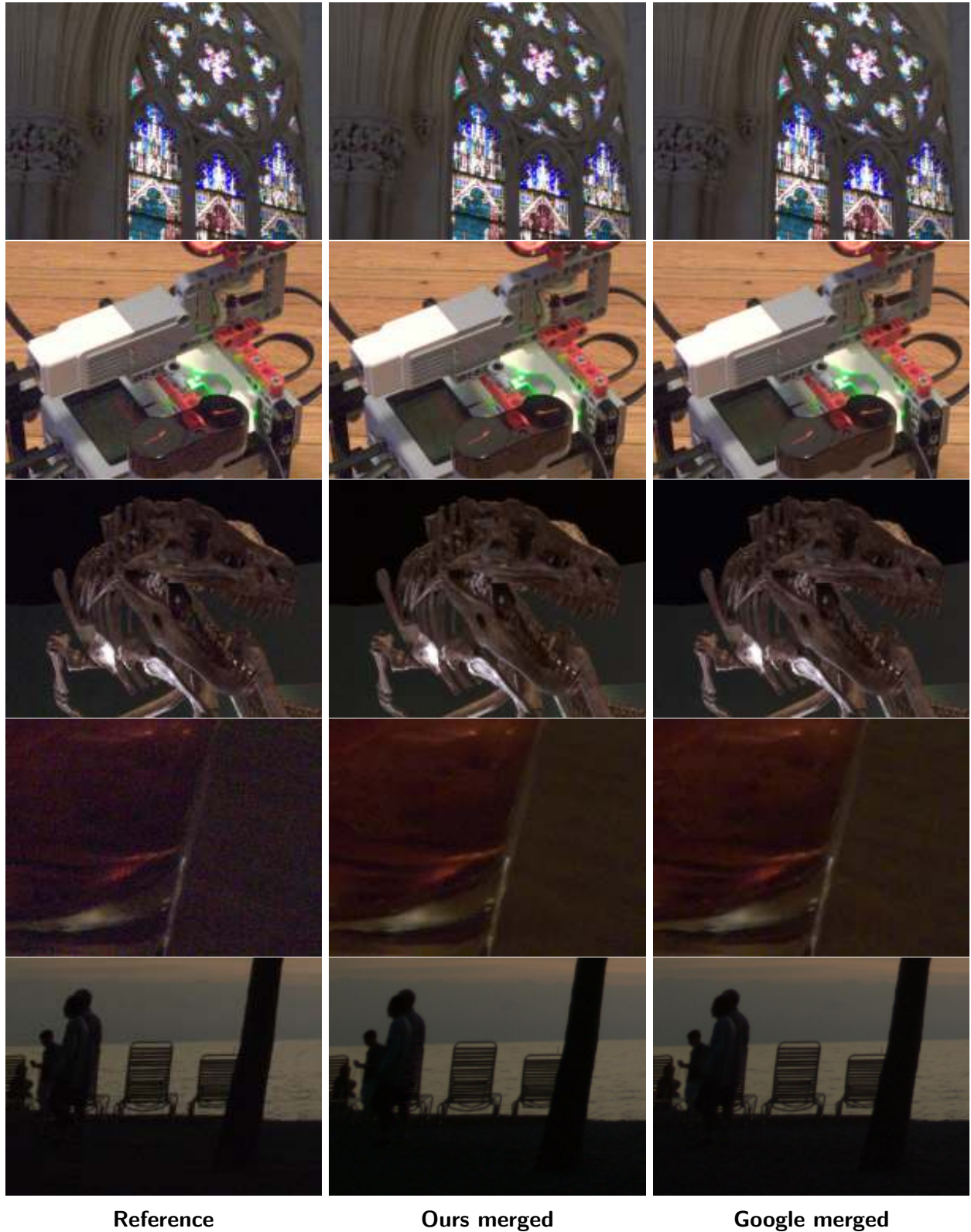
---

[14] https://photos.app.goo.gl/mEhyNrqKc2x7rbqj9

Reference   **Ours merged**   **Google merged**

Figure 13: Visual comparison of crops of minimally processed versions of the reference image, our "merged" result and Google's "merged" result (viewers are invited to zoom in). Both alignment and merging procedures significantly reduce noise. *Top row*: both merged results are virtually indistinguishable. *Second row*: Google seems to further reduce noise. *Third row*: our result seems more denoised. *Fourth row*: Google's residual noise seems to be of lower spatial frequency compared to ours (they might be using a larger tile size for this low-light image). *Bottom row*: for this image, ghosting is slightly stronger in our result.

6.3   Visual Comparison to Google's Final Results

The presence of their final results also enables visual comparison of their consumer-grade raw to JPEG finishing pipeline against our own. That said, given the simplified nature of our own finishing step, if we compare our final images to Google's cleverly chroma denoised, tone-mapped, sharpened, dehazed, color corrected (and so on and so forth) final HDR+ results, such comparison is definitely not in our favor in the vast majority of cases. It would certainly make sense to fully implement these additional finishing steps, but that would require a lot of additional development, testing and fine-tuning time to reach a similar level of quality. The added complexity would also likely imply longer processing times in our Python implementation, and would stir this article away from our main subject of focus, the raw burst alignment and merging algorithm. Still, if we compare our images to a simple post-processing of the reference image (demosaicking + white balance + color matrix + gamma curve), we can see that our pipeline does provide significant visual improvement. Some examples are featured in Figure 14. Final full-size results for the whole subset of 153 bursts can be consulted and downloaded from another Google Photos album[15].

7   Results on GoPro Bursts

In order to further verify the appeal of the HDR+ raw burst denoising algorithm and the improvements provided by parts of its finishing pipeline, we set out to test our own implementation on bursts that were not part of the HDR+ dataset. We used GoPro HERO 8 Black and Hero 9 Black action cameras as an alternative to the Android devices used by Google. They feature similar sensors (Sony CMOS sensor with relatively small pixels) but much wider lenses when compared to the main camera typically found in modern smartphones.

We captured sequences of 10 raw images using the dedicated burst mode of the HERO 8 and 9, all while trying to follow the base principles described in [9] (underexposed bursts, with identical focal length and short exposure time for all images, and scene and camera motion that is manageable with respect to exposure time). Here are the HERO9 parameters we had access to in order to respect these principles as much as possible:

- We did not have direct control of the exposure time, as it is handled by the camera's auto-exposure algorithm. Instead, we could specify the burst rate (how many images we want in a specified amount of time). We set it to its fastest value (10 images in 1s when capturing raw images), ensuring a maximum exposure time of 0.1 s.

- We could also change the EV compensation parameter (which controls how bright the burst images are). Since we wanted the bursts to be underexposed, we typically set it between 0 and -2 depending on scene brightness, selected ISO and the expected behavior of the auto-exposure algorithm.

- Finally, we could set minimum and maximum ISO values. We actually had to set both parameters to the same value if we wanted all images of the burst to feature the same ISO (if we didn't, the AE algorithm might have selected different values for different images of the same burst). This means that we had to manually pick ISO before capture. We usually set ISO to 100 in most scenarios (in order to avoid clipped highlights) except in low light where we might set it between 200 and 800.

We deliberately shot most bursts in very low light / high dynamic range scenarios to ensure these would have a lot of noise. Since GoPro cameras output raw images in their proprietary `.gpr` format,

---

[15]https://photos.app.goo.gl/QurTFcvUMc8i1DM7A

Reference   **Ours final**   **Google final**
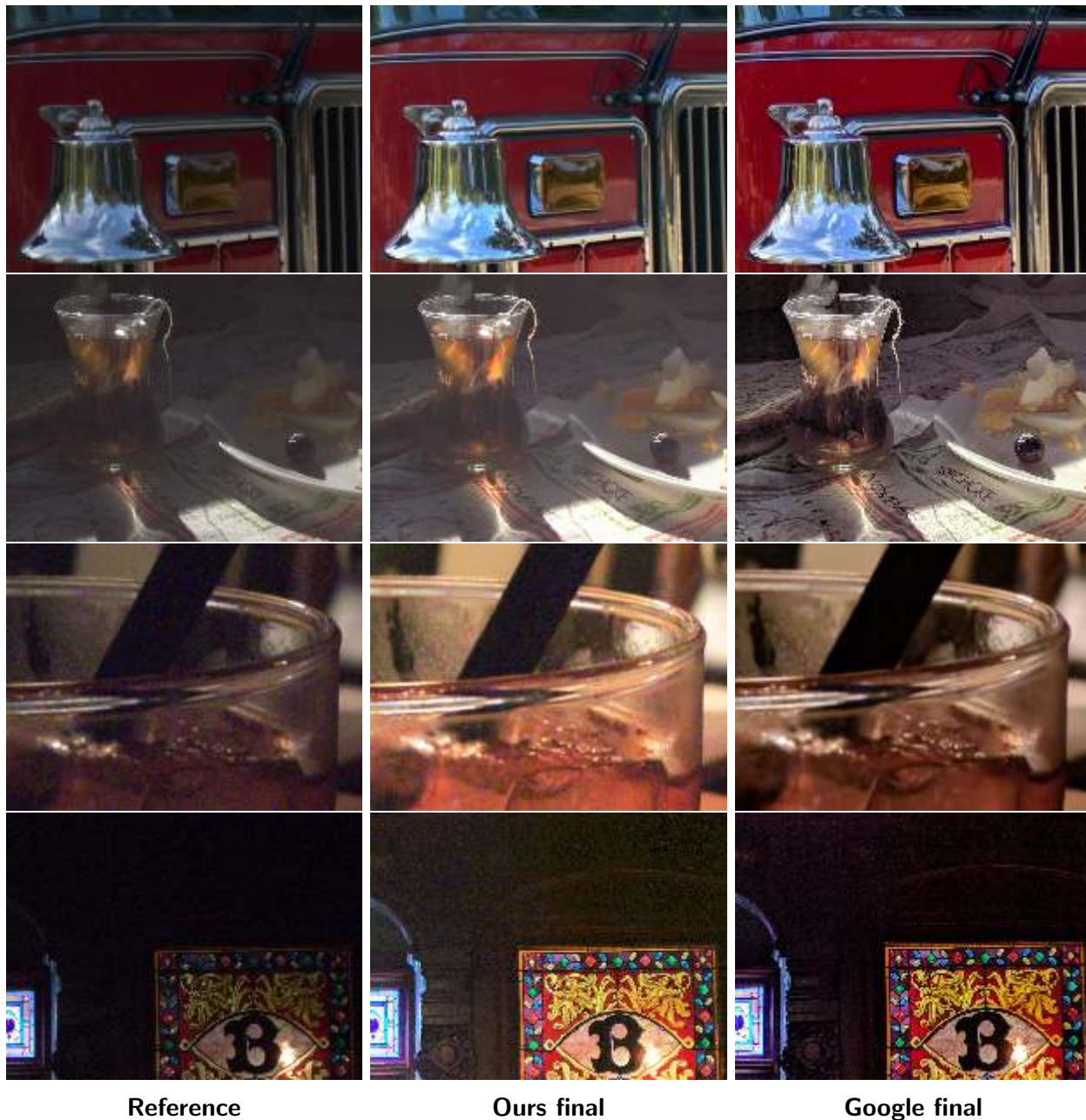
Figure 14: Visual comparison of crops of the minimally processed reference image, our final result and Google's final result (viewers are invited to zoom in). Google's different spatial denoising, tone mapping, sharpening and additional operations (chroma denoising, lens shading correction, dehazing, color adjustment, dithering, etc) ensures better results in almost any scenario. *First Row*: our results can be similar to Google's in some static, well lit environments. *Second row*: Google's more aggressive tone mapping produces an image with better dynamic range at the expense of more residual noise. *Third row*: since we don't apply any chroma denoising, residual noise can be amplified by our tone mapping and sharpening operations. *Bottom row*: Google also seems to perform hot pixel removal, which we do not (look at the top right corner of the crops).

we converted them to Adobe Digital Negative (`.dng`) files using Adobe DNG Converter[16], which is the same format used for the HDR+ dataset. We then simply executed the exact same code we ran on the images provided by Google, without changing any tuning parameter.

We applied our own finishing pipeline (described in Section 5.2) to both the reference image and the result of the alignment and merging steps in order to assess their impact on noise reduction. We also added a minimally processed version of the reference image (with no tone-mapping) to show how bright the original raw image actually is. We can see that noise is significantly reduced by the alignment and merging procedure (even on image edges, where these steps might perform worse because of image distortion induced by the wide lens). Residual noise can however still be quite present in some images (particularly in very low light), and some of it can be amplified by the tone mapping and sharpening steps of our pipeline. Figure 15 showcases some of these results, while the full-size results on 17 GoPro bursts can be consulted and downloaded from Google Photos[17]. The 17 raw DNG bursts can also be downloaded from Google Drive[18].

---

[16]https://helpx.adobe.com/photoshop/using/adobe-dng-converter.html
[17]https://photos.app.goo.gl/XKiiSSNmqUEMYpYS8
[18]https://drive.google.com/drive/folders/1j2NIEPSnrdjSOsjL1kzl3VEmYKBkChJD?usp=sharing

Reference (minimal processing)   **Reference + finishing** pipeline   **Merged + finishing** pipeline
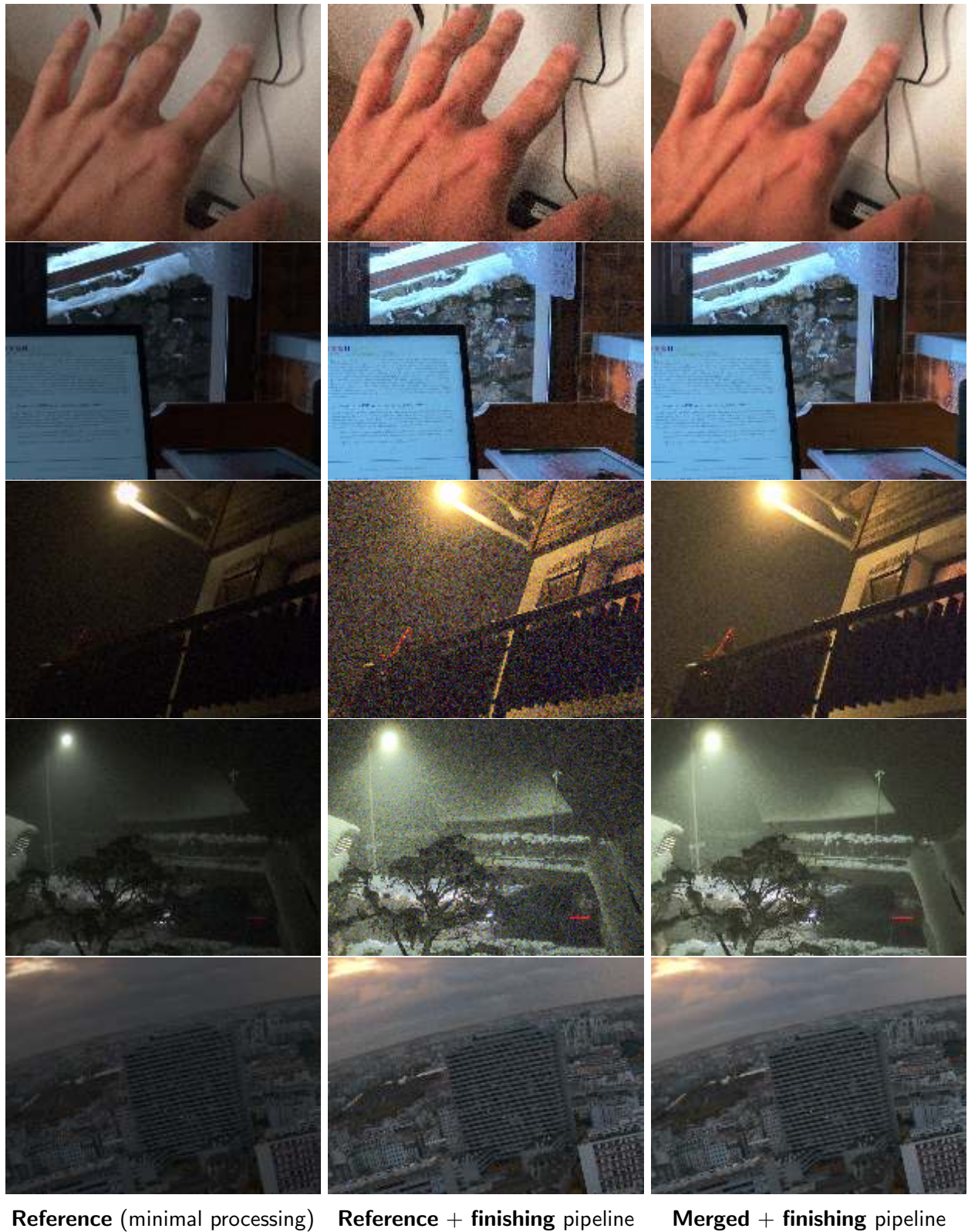
Figure 15: Visual comparison of crops of the minimally processed reference image, the reference processed by our finishing pipeline, and our alignment and merging result processed by our finishing pipeline (viewers are invited to zoom in).

8 Conclusion

Throughout this paper, we presented the core principle at the heart of the HDR+ digital photography pipeline: a raw burst alignment and merging algorithm. It uses a Wiener filter variant to combine the information of aligned tiles stacks in the 2D DFT space, to produce individual tiles with significantly less noise. Little priors and image metadata are required to produce convincing results, the most important one being a crude estimation of the noise level. When tuned properly, a good compromise between residual noise and ghosting artifacts can be found, resulting in natural, visually pleasing images. Many additional steps are required after alignment and merging to obtain a final RGB image acceptable for consumer-grade products, especially when trying to stand out from competitors. When compared to more recent state-of-the art approaches, the constrained computational complexity of the whole pipeline makes it well suited for mobile photography, but the strategies employed are versatile enough to improve image quality on many different types of cameras.
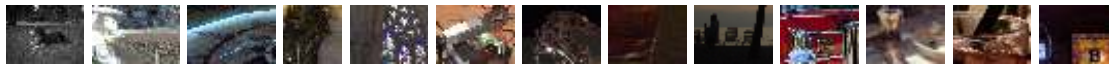
Acknowledgment

Image Credits

 Captured by the authors using GoPro Hero 8 and Hero 9 cameras

 Extracted from the HDR+ Burst Photography Dataset

References

[1] J. Anger and E. Meinhardt-Llopis, *Implementation of Local Fourier Burst Accumulation for Video Deblurring*, Image Processing On Line, 7 (2017), pp. 56–64. https://doi.org/10.5201/ipol.2017.197.

[2] T. Brooks, B. Mildenhall, T. Xue, J. Chen, D. Sharlet, and J.T. Barron, *Unprocessing images for learned raw denoising*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 11036–11045. http://dx.doi.org/10.1109/CVPR.2019.01129.

[3] A. Buades, Y. Lou, J-M. Morel, and Z. Tang, *A note on multi-image denoising*, in International Workshop on Local and Non-Local Approximation in Image Processing, IEEE, 2009, pp. 1–15. http://dx.doi.org/10.1109/LNLA.2009.5278408.

[4] M. Colom and A. Buades, *Analysis and extension of the percentile method, estimating a noise curve from a single image*, Image Processing On Line, 3 (2013), pp. 332–359. http://dx.doi.org/10.5201/ipol.2013.90.

[5] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, *BM3D Image Denoising with Shape-Adaptive Principal Component Analysis*, in SPARS'09 - Signal Processing with Adaptive Sparse Structured Representations, Rémi Gribonval, ed., Saint Malo, France, Apr. 2009, Inria Rennes - Bretagne Atlantique. https://hal.inria.fr/inria-00369582.

[6] M. Delbracio and G. Sapiro, *Removing camera shake via weighted fourier burst accumulation*, IEEE Transactions on Image Processing, 24 (2015), pp. 3293–3307. http://dx.doi.org/10.1109/TIP.2015.2442914.

[7] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian, *Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data*, IEEE Transactions on Image Processing, 17 (2008), pp. 1737–1754. http://dx.doi.org/10.1109/TIP.2008.2001399.

[8] B.K. Gunturk, J. Glotzbach, Y. Altunbasak, R.W. Schafer, and R.M. Mersereau, *Demosaicking: color filter array interpolation*, IEEE Signal processing magazine, 22 (2005), pp. 44–54. http://dx.doi.org/10.1109/MSP.2005.1407714.

[9] S.W. Hasinoff, D. Sharlet, R. Geiss, A. Adams, J.T. Barron, F. Kainz, J. Chen, and M. Levoy, *Burst photography for high dynamic range and low-light imaging on mobile cameras*, ACM Transactions on Graphics (Proceedings SIGGRAPH Asia), 35 (2016). http://dx.doi.org/10.1145/2980179.2980254.

[10] G.E. Healey and R. Kondepudy, *Radiometric CCD camera calibration and noise estimation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16 (1994), pp. 267–276. http://dx.doi.org/10.1109/34.276126.

[11] C. Hessel, *An Implementation of the Exposure Fusion Algorithm*, Image Processing On Line, 8 (2018), pp. 369–387. https://doi.org/10.5201/ipol.2018.230.

[12] ——, *Simulated Exposure Fusion*, Image Processing On Line, 9 (2019), pp. 469–482. https://doi.org/10.5201/ipol.2019.279.

[13] K. Hirakawa and T. W. Parks, *Adaptive homogeneity-directed demosaicing algorithm*, IEEE Transactions on Image Processing, 14 (2005), pp. 360–369. https://doi.org/10.1109/TIP.2004.838691.

[14] M. Lebrun, A. Buades, and J-M. Morel, *A nonlocal Bayesian image denoising algorithm*, SIAM Journal on Imaging Sciences, 6 (2013), pp. 1665–1688. http://dx.doi.org/10.1137/120874989.

[15] O. Liba, K. Murthy, Y-T. Tsai, T. Brooks, T. Xue, N. Karnad, Q. He, J.T Barron, D. Sharlet, R. Geiss, S.W. Hasinoff, Y. Pritch, and M.S. Levoy, *Handheld mobile photography in very low light*, ACM Transactions on Graphics (TOG), 38 (2019), pp. 1–16. http://dx.doi.org/10.1145/3355089.3356508.

[16] T. Mertens, J. Kautz, and F. Van Reeth, *Exposure fusion: A simple and practical alternative to high dynamic range photography*, in Computer Graphics Forum, vol. 28, Wiley Online Library, 2009, pp. 161–171. http://dx.doi.org/10.1111/j.1467-8659.2008.01171.x.

[17] B. Mildenhall, J.T. Barron, J. Chen, D. Sharlet, R. Ng, and R. Carroll, *Burst denoising with kernel prediction networks*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2502–2510. http://dx.doi.org/10.1109/CVPR.2018.00265.

[18] M. Rakhshanfar and M. A. Amer, *Estimation of Gaussian, Poissonian-Gaussian, and Processed Visual Noise and Its Level Function*, IEEE Transactions on Image Processing, 25 (2016), pp. 4172–4185. http://dx.doi.org/10.1109/TIP.2016.2588320.

[19] K. Zhang, W. Zuo, and L. Zhang, *FFDNet: Toward a fast and flexible solution for CNN-based image denoising*, IEEE Transactions on Image Processing, 27 (2018), pp. 4608–4622. http://dx.doi.org/10.1109/TIP.2018.2839891.