Fast Temporal Reprojection without Motion Vectors

Johannes Hanika
Karlsruhe Institute of Technology

Lorenzo Tessari
Karlsruhe Institute of Technology

Carsten Dachsbacher
Karlsruhe Institute of Technology

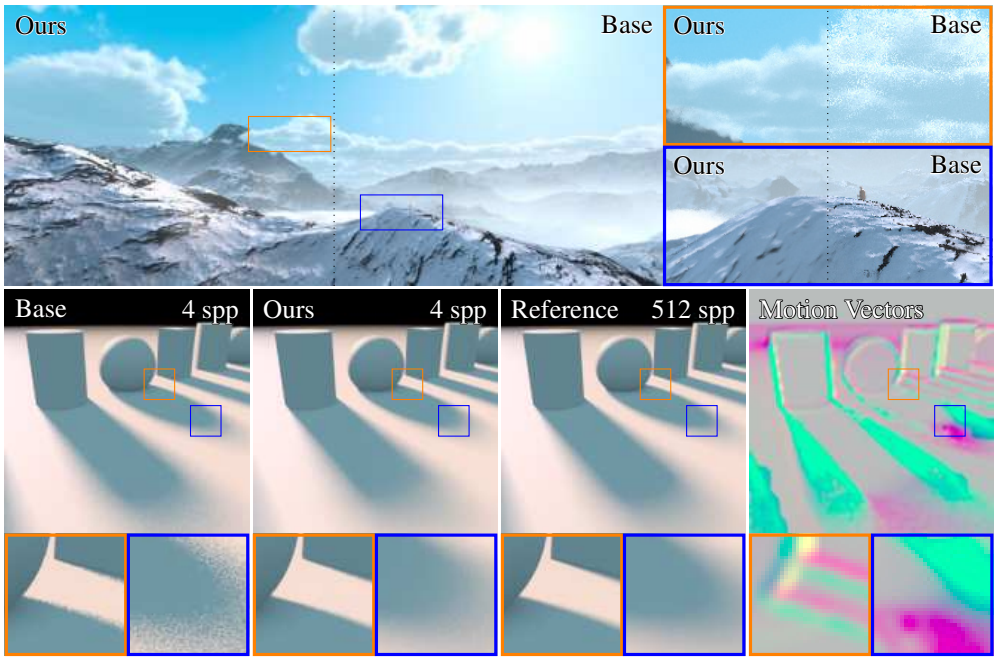

Figure 1. Top: The HIMALAYA scene, rendered with (left) and without (right) reprojected information from previous frames. The procedural scene is composed of ray-marched signed distance fields and volumetric, scattering clouds. The clouds move due to complex, nonlinear noise functions, and the features seen on screen cannot be mapped to a unique depth coordinate due to semitransparency. This makes estimation of true motion vectors a hard problem or, at times, even impossible. Bottom: The PENUMBRA scene, where a moving light source causes the shadow to move separately from the geometry. We devise a fast algorithm to compute image-space correspondences that can be used as motion vectors in various reprojection scenarios, such as temporal denoising.

 ISSN 2331-7418

Abstract

Rendering realistic graphics often depends on random sampling, increasingly so even for real-time settings. When rendering animations, there is often a surprising amount of information that can be reused between frames.

This is exploited in numerous rendering algorithms, offline and real-time, by relying on reprojecting samples, for denoising as a post-process or for more time-critical applications such as temporal antialiasing for interactive preview or real-time rendering. Motion vectors are widely used during reprojection to align adjacent frames' warping based on the input geometry vectors between two time samples. Unfortunately, this is not always possible, as not every pixel may have coherent motion, such as when a glass surface moves: the highlight moves in a different direction than the surface or the object behind the surface. Estimation of true motion vectors is thus only possible for special cases. We devise a fast algorithm to compute dense correspondences in image space to generalize reprojection-based algorithms to scenarios where analytical motion vectors are unavailable and high performance is required. Our key ingredient is an efficient embedding of patch-based correspondence detection into a hierarchical algorithm. We demonstrate the effectiveness and utility of the proposed reprojection technique for three applications: temporal antialiasing, handheld burst photography, and Monte Carlo rendering of animations.

1. Introduction

Generating believable computer-generated imagery involves the highest requirements on all components, because the human eye is extremely critical. In particular, it requires one to simulate photorealistic transport of light to generate familiar global illumination in detailed and complex scenes. This means that many effects can only be created using Monte Carlo sampling, for instance, by using full path tracing or by computing individual effects such as soft shadows [Pharr et al. 2016]. In very detailed scenery, even antialiasing of a geometry can result in noisy renderings.

To address this issue, usually many hundreds or thousands of samples are generated per pixel. Even for offline rendering applications, it is common practice to denoise the resulting images once Monte Carlo sampling passes the line of diminishing returns for computational cost versus quality obtained [Zwicker et al. 2015]. This strategy becomes increasingly more important when faster frame times are required, such as for interactive preview rendering for look development or layout or for real-time rendering for games.

Such noise reduction techniques usually work spatially (i.e., inter-pixel) and often have a temporal component (i.e., inter-frame): the previous (and/or next) frames are warped to approximately align with the current frame to generate a larger amount of effective samples per pixel via reprojection.

This can be done precisely by retracing visibility rays [Fascione et al. 2019; Bekaert et al. 2002]. Inspired by the availability of fast ray tracing hardware (NVIDIA's RTX and the vendor-neutral Khronos extension to the Vulkan API), we are mostly in-

Figure 2. Examples of complex nonlinear motion. Top: The cloud front is formed and evolves as 2D coverage noise and 3D Worley noise interact (see [Tatarchuk et al. 2015]). Bottom: Multiple wave patterns with different velocities interfere and specular highlights move in an emergent direction. Analytically deriving motion vectors to warp a previous frame into the next one is extremely hard and often not possible in such cases.

terested in real-time performance. We want to empower applications like Spatiotemporal Variance-Guided Filtering (SVGF) [Schied et al. 2018], where Schied et al. spent substantial effort deriving path-space gradients for indirect lighting, by providing fast image-space gradients to align the previous frame.

To provide visually-rich geometric detail, procedural modeling and physics simulation for volumetrics and fluids are indispensable tools. Unfortunately, it can be close to intractable to derive motion vectors in such cases (see Figure 2).

Fluids are often simulated using particles, from which a smooth surface is reconstructed via implicits such as *blobbies* [Blinn 1982]. The movement of the resulting surface is not trivially expressed by the motion vectors of the input particles [Stam and Schmidt 2011]. Even worse, for instance, an ocean surface will have foam at the tips of the waves. The waves are formed by the particles, but these do not move with the wave ridge, usually rotating underneath. For a noise reduction application, however, we would like to track the salient features with similar shading, i.e., the diffuse foam on the wave ridge, or even a specular reflection. The world-space motion vectors of the particles are thus rendered useless. Similar concerns apply to ocean rendering via Fourier synthesis, signed distance fields, and procedural volumetrics. Even explicit tessellation using marching cubes does not solve the issue because of inter-frame topology changes. When rendering clouds, the volumetric density is continuously changing between frames, and the evolution cannot be described by rigid motion. This makes it impossible to obtain world-space motion vectors, yet we can still match features in 2D image space. Another difficult scenario arises when tracking radiance from secondary effects such as subsurface scattering or shadows with moving lights: in the first case the pattern changes in non-rigid ways, making motion estimation unreliable, whereas in the second case each light source would have to be tracked separately. We follow the observation that estimating true motion vectors is an ill-posed problem [Buades et al. 2005a] and pursue the search for relevant pixel correspondences instead. Previous work has used multi-scale pyramids and reg-

ularized optical flow methods for robust patch-based estimation [Buades and Lisani 2020]. We use a multi-scale approach to achieve real-time performance. To make use of reprojected frames in real time, we compute a dense set of pixel correspondences, which satisfy a number of properties. Our algorithm is fast, is resilient against noise, and aligns similar colors rather than estimating plausible motion.

In summary, our contribution is a fast method to compute hierarchical pixel correspondences based on *non-local means* (NLM), which is a denoising technique that searches similar pixels by matching the contents of the surrounding patches [Buades et al. 2005a]. To make this efficient, we use an intermediate warping step between the levels of the hierarchy, along with a subpixel offset estimation step to reduce the jitter otherwise caused by the hierarchical approach. This subpixel estimation can also be used to provide fine-scale offsets while operating on a downsampled buffer only, further speeding up the operation. Our approach is general and, as such, can extract motion where other methods struggle, and it is suitable when real-time interaction is essential. We demonstrate the performance of the resulting algorithm on three applications (temporal antialiasing, burst photography, and Monte Carlo rendering). Full source code is available.

2. Previous Work

This paper is inspired by work in computational photography [Hasinoff et al. 2016; Wronski et al. 2019; Liba et al. 2019]. In this field, there are similar performance constraints and resulting trade-offs with respect to alignment quality. Misalignment is corrected by rejecting some pixels, much like it is done in rendering.

Optical flow. We do not attempt an exhaustive comparison to the extensive existing optical flow literature (see, e.g., [Butler et al. 2012]) because our aim is different than that of the vision community (see [Buades et al. 2005a]). We want to strike different trade-offs, mainly less quality and more speed, but also we do not require plausible motion but a warped picture that has RGB values as close as possible to the reference image. We evaluate this criterion against a selection of optical flow techniques [Farnebäck 2003; Sánchez Pérez et al. 2013; Plyer et al. 2016] in Section 5. In particular, the coarse-to-fine hierarchical Lucas–Kanade approach of eFOLKI [Plyer et al. 2016] is related to ours. We replace the iterative inner core of the method by a simpler non-iterative technique related to non-local means (see the end of this section). Our hierarchical scheme is closely related to the iterative warping scheme that has been applied to optical flow [Le Besnerais and Champagnat 2005] but has been shown to result in divergent behavior in conjunction with the iterative core of the method. We show that in our context, the scheme is stable.

Reprojection. Reprojection has been proposed many times in different contexts. For instance, Bekaert et al. [Bekaert et al. 2002] reuse paths for neighboring pixels, which is a variation of reuse over time [Fascione et al. 2019]. Temporal reprojection has been explored for shading and for shadows [Nehab et al. 2007; Scherzer et al. 2007]. We explore simple image-based reprojection, without precisely looking at the transport paths.

Temporal antialiasing. One of our applications is temporal antialiasing (TAA). We refer to a recent survey [Yang et al. 2020] for an extensive discussion of the state of the art. TAA is usually performed by using known motion vectors of the input geometry. Extending this to more general cases such as indirect lighting, complex motion, or transparency is a hard problem and requires special case treatment [Olejnik and Kozłowski 2020].

General correspondences. A fast CPU method to compute general correspondences between pixels in adjacent frames is *patch match* [Barnes et al. 2009]. This iterative method converges impressively fast at the beginning, due to the collaborative search where results are shared between pixels. In our implementation we had issues converging to final quality, however, and concluded that other approaches might be more suitable for a very fast GPU implementation. This would be needed to replace the gradient computation in real-time denoising approaches, such as Adaptive Spatiotemporal Variance Guided Filtering (ASVGF) [Schied et al. 2018].

There is work on computing motion vectors for indirect lighting effects for offline rendering [Zimmer et al. 2015]. This approach can track the movement of highlights via manifold walks, motion vectors, and geometric derivatives. It will, however, compute precise screen-space motion for every effect depending on path length. This does not lend itself well for an image-based denoising pipeline, and it suffers from the curse of dimensionality (because every bounce needs to be processed separately). It also uses a non-local means approach for the noise removal step. Though we do not propose any new method for denoising, we employ the core of the non-local means algorithm in the core of our correspondence estimation.

Non-local means. NLM is a method originally devised for image denoising [Buades et al. 2005b]. It works by match-making between pixels by looking at a patch of context pixels around every pixel and comparing the difference between the two patches. Similar pixels are averaged and written back to yield an image with less noise. This approach has been extended to multiple frames and has been used to estimate motion vectors for ray tracing of implicits [Roegner et al. 2015]. However, the authors did not attempt a real-time implementation. Today, we can embed such an estimation into more mature temporal denoising methods, and also there is a more pressing need due to the widespread availability of hardware ray tracing. We exploit a technique to

speed up non-local means in a GPU implementation [Darbon et al. 2008] and extend it to hierarchical matching and subpixel offset estimation.

3. Computing Fast Pixel Correspondences

We take as input a new noisy image and another image (noisy or the result of previous runs) to align to the first image.

The search for corresponding pixels between the frames is expensive, and the cost increases with the size of the search window around each pixel. In order to efficiently find matching pixels at greater distances, we use an iterative scheme that works hierarchically on a resolution pyramid. This allows us to extend a small window to a larger effective search radius: on each hierarchy level we want to compute the best displacement vector for each pixel akin to non-local means, which compares the (weighted) differences of two pixel neighborhoods. A naive approach to this is too costly; however, accelerated variants compute pixel differences for the whole buffer at once (Section 3.1) and use the exact same search window for every pixel. This is fundamentally incompatible with hierarchical processing, where every pixel has a different input from the coarser level. To overcome this problem, we introduce a warping scheme that iteratively aligns the coarse images before refining the displacement vectors on the next-finer level. After alignment, we use the residual patch distances to drive rejection schemes to remedy alignment errors (Section 3.2).

3.1. Alignment

We make use of a non-local means pattern-matching scheme and iterate it in a hierarchical coarse-to-fine manner. All our processing is done on grayscale images. A schematic overview of one such step at a single scale is depicted in Figure 3.

Traditional non-local means finds the shift $s_k \in S$ to similar pixels in the neighborhood S of a pixel (i, j) by comparing a local patch $p \in P$ around (i, j) and computing a patch-based distance by summing the pixel distances weighted by $w(p)$:

$$d_k = \sum_{p \in P} w(p) \cdot \left\| I\left((i,j)+p\right) - I\left((i,j)+s_k+p\right) \right\|^2. \tag{1}$$

Implementing this in a straightforward manner results in three nested loops:

```
for each pixel (i,j) in the image I
  for each shift s_k in a search window S
    d_k = 0
    for each p in a patch P
      d_k += w(p)*|I((i,j)+p) - I((i,j)+s_k+p)|^2
```

This has an asymptotic runtime of $O(|I||S||P|)$. Here, w are some arbitrary smoothing weights depending on the patch offset p (this can be Gaussian or a box blur).

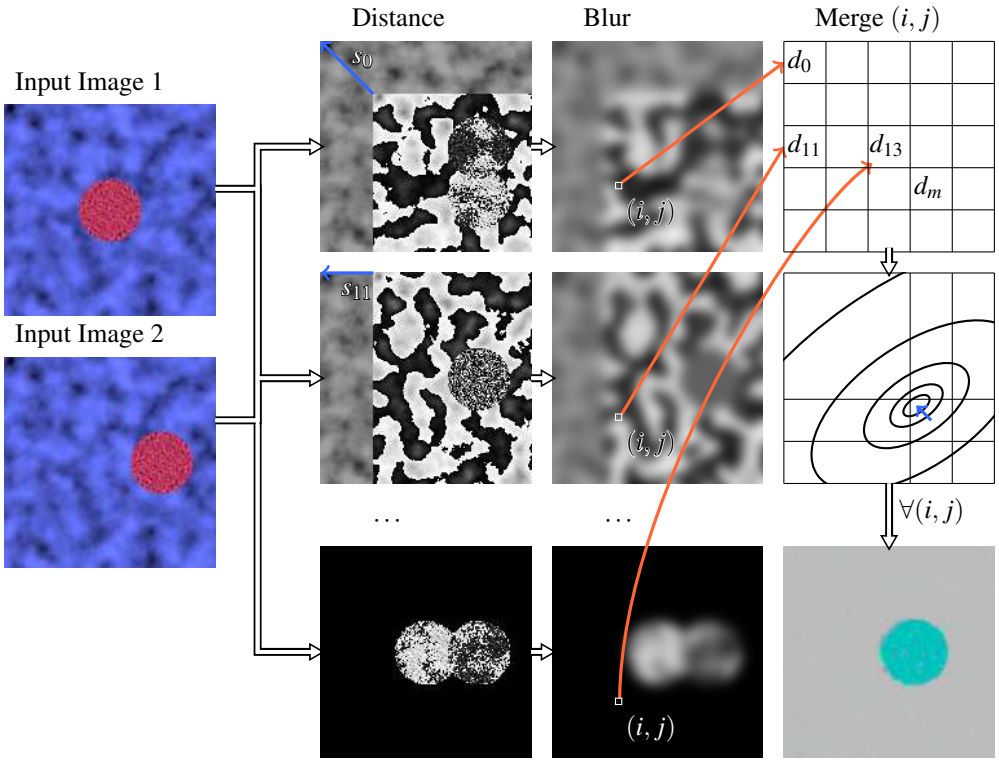

Figure 3. Pixel correspondence estimation on one scale (see Section 3.1). The two input images (left) pass through three stages: *distance*, *blur*, and *merge*, indicated by the three columns. First, the images are offset by the set of 25 shift vectors $s_k \in [-2, 2]^2$, represented by the rows in the *distance* and *blur* columns. The output of the blur stage represents the patchwise distance between the two images for the given pixel (i, j) and shift s_k. The *merge* stage collects all 25 of these distances d_k and selects the minimum d_m. Around this position in the 5×5 stencil of collected distances, a quadric is computed, which points us to a subpixel motion vector. The end result is s_m plus this subpixel shift.

One optimization that significantly speeds up the computation on one level in the hierarchy is to reorder the computation and evaluate the shifts $s_k \in S$ for all pixels simultaneously [Darbon et al. 2008]:

```
for each shift s_k in a search window S
  for each pixel (i,j) in the image I
    d_k(i,j) = |I(i,j) - I((i,j)+s_k)|^2
  convolve d_k(i,j) by w(p)
```

This convolution can be implemented by any means of a fast blur kernel. This scheme avoids duplicate computation of pixel distances, has better asymptotic runtime $O(|S||I|)$, and results in more cache-friendly memory access patterns. It also depends on a standard convolution that can be optimized aggressively, as the kernel

shape may be approximated. The exact shape of $w(p)$ does not impact the results drastically, so an $O(|I|)$ box blur can be used.

Our main contribution is to efficiently embed this scheme into a hierarchical algorithm: Because the runtime depends on the search window size $|S|$, we use a small $S = [-2,2]^2$ and build a hierarchical pipeline around it to extend the maximum offset. We split the alignment process into four kernels: *distance* computation, *blur*, *merge* (Figure 3), and *warp* for the hierarchical application (Figure 4).

Distance kernel. To compute the L_2 distance for all shift offsets $s_k \in S$, we use a distance kernel, taking two grayscale images I_1 and I_2 as input. For every $s_k \in S$, we offset the two images, compute the L_2 distance $d_k(i,j)$ between two pixels $I_1(i,j)$ and $I_2((i,j)+s_k)$, and output it in an array of $|S|$ images.

Blur kernel. We then blur this array of images using a 3D compute shader dispatch, running over the dimensions of the image and the length of the array (width, height, $|S|$). Because any blur kernel can be used, we employ an iterated filter that approximates a Gaussian shape and downsamples the intermediate buffers to save memory bandwidth (similar to Kawase blur [Kawase 2003]). We make use of hardware texture interpolation to simulate a 5×5-tap filter using five texture fetches. Note that this scheme is strictly speaking not $O(|I|)$ as a box blur would be, but is a lot faster on GPU than such an implementation.

Merge kernel. The merge kernel takes an array of $|S|$ blurred distance images as input. One pixel in the kth image represents the patch-based distance between the images I_1 and I_2 offset against each other by the shift vector s_k. For every pixel, we find the offset $s_k \in S$ that results in the minimum patch distance. This amounts to searching through the $|S|$ distance images and remembering the index k with the smallest value.

However, because this is a discrete decision, it will only output integer shift vectors, which makes subtle subpixel camera movements jittery and causes problems with hierarchical processing. Thus, we insert another step to estimate subpixel shift vectors. We employ a method similar to one used for burst photography [Hasinoff et al. 2016] and utilize it at several levels in the hierarchy to reduce the propagation of errors: First, the $|S|$ distance values d_k are written next to each other in a grid, according to the corresponding shift vectors s_k (see Figure 3, top right). Then, we select the minimum value $d_m \leq d_k \forall k$. We next compute a quadric matching the 3×3 neighborhood of d_m (Figure 3, center right) and compute its minimum to arrive at an interpolated subpixel offset (marked in blue in the figure). We skip this step on the coarsest two levels of hierarchical processing because the results were prone to aliasing.

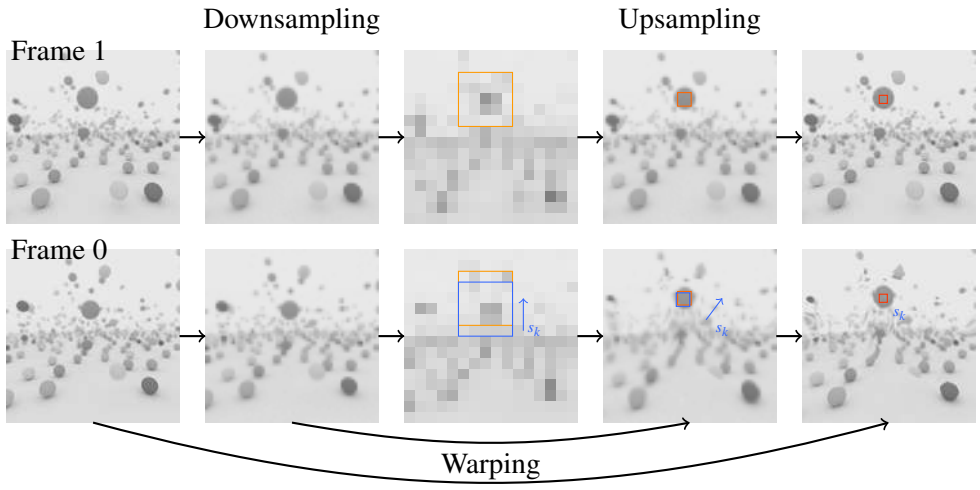

Figure 4. Illustration of the hierarchical warping process. The two frames are aligned coarse to fine by iteratively matching patches around each pixel to determine shifts s_k on every level $l = 3, \ldots, 0$ (only three levels shown here). After matching, frame 0 is warped to align with the shifts on this level, and the residual difference to frame 1 on the next level is searched for. For illustration purposes, the frames are actually ten frames apart to emphasize the differences. This causes fail cases; see, for instance, the white sphere in the bottom right corner in the foreground.

Hierarchical processing. To extend the effective search window beyond $|S|$, we iterate the previous three kernels in a hierarchical fashion. The input is first downsampled a few times, and then the three alignment kernels are run on each level l, starting at the coarsest. The next-finer level l receives the shift vectors s_k per pixel from the coarser level $l+1$ as input. Because the distance kernel as previously described needs to compute the same offset s_k for all pixels in the image, it cannot support summing up such coarse shift vectors varying per pixel. To work around this, we conceptually introduce a warp kernel, which pre-warps the coarse input images coming from level $l+1$. On level l we only need to search for fine displacement offsets that will be summed to the coarse ones. The processing graph is visualised in Figure 4. This illustration uses only three resolution levels. In our implementation we use four, at a reduction of $4\times$ in both dimensions. This results in $\sum_{l=0}^{3} \pm 2 \text{ px} \cdot 4^l = \pm 170$ px maximum offset. In practice we do not use a separate compute shader dispatch for the warp kernel but input the coarse offsets into the distance kernel directly.

Discussion. Warping changes the results algorithmically as compared to other hierarchical, patch-based similarity matching algorithms. It replaces the square neighborhood on all but the coarsest levels by a warped context window around the current pixel. Because the square context region was only a heuristic choice, this has no neg-

ative impact on the result. We use bilinear interpolation to read the coarse offsets when processing the next-finer image. This prevents our warped context windows from taking on extreme deformed shapes. Because our algorithm is non-iterative, we also do not suffer from divergence issues such as previous hierarchical schemes [Le Besnerais and Champagnat 2005].

3.2. Blending/Rejection

Ideally we would want zero alignment error and very good agreement of the new frame buffer and the warped one. Unfortunately this is not always possible, due to disocclusions and sometimes an overaggressive implicit smoothing constraint that is enforced by the hierarchical matching scheme. Thus, we make use of the patch distance that is computed on the finest level of the alignment process, to filter out misaligned pixels. In particular, we choose the blend weight $\alpha' = \text{clamp}(\alpha \cdot (1-m), 0, 1)$. Here, α is the user-supplied blend weight and m is a mask that is computed from the patch distance ε as $m = \kappa \cdot \varepsilon - \eta$, where κ scales the error up and η subtracts a noise threshold.

To make motion vector detection more robust, we compute the distance for a shift d_k in a few different variants that are useful for different applications. In particular we experimented with the following:

$$L_2: \quad d_k = \left|I(i,j) - I\left((i,j)+s_k\right)\right|^2 \tag{2}$$

$$L_1: \quad d_k = \left|I(i,j) - I\left((i,j)+s_k\right)\right| \tag{3}$$

$$\log L_1: \quad d_k = \log\left(2 + \left|I(i,j) - I\left((i,j)+s_k\right)\right|\right) \tag{4}$$

$$\log L_1^c: \quad d_k = \log\left(2 + \left|I(i,j) - I\left((i,j)+s_k\right)\right|\right) \cdot \left(2 - \exp(-0.02|s_k|^2)\right). \tag{5}$$

To get cleaner motion vectors, we regularized the L_1 distance by using a center weighting (i.e., the exponential part in Equation (5)). Because this weight is in the range of $[1,2]$ (it only increases the distance when the shift s_k is large), we compress the L_1 norm into a similar range. It worked well in our tests to use a simple logarithmic transform here. The effect on both error masks and motion vectors can be observed in Figure 5.

3.3. Parameters

Our technique has a few parameters. The first set consists of the blur radii on every scale. For relatively noise-free image sequences, a constant radius of 2 works well because it results in a 5×5 neighborhood that captures salient features in full HD resolution. Figure 1 has been computed with this setting, for instance. The trade-off for larger radii is the same as for non-local means (see Figure 6): larger patch sizes are more robust to noise but result in less sharpness. The influence of the patch distance from the alignment process on rejection is steered by the two parameters κ and η.

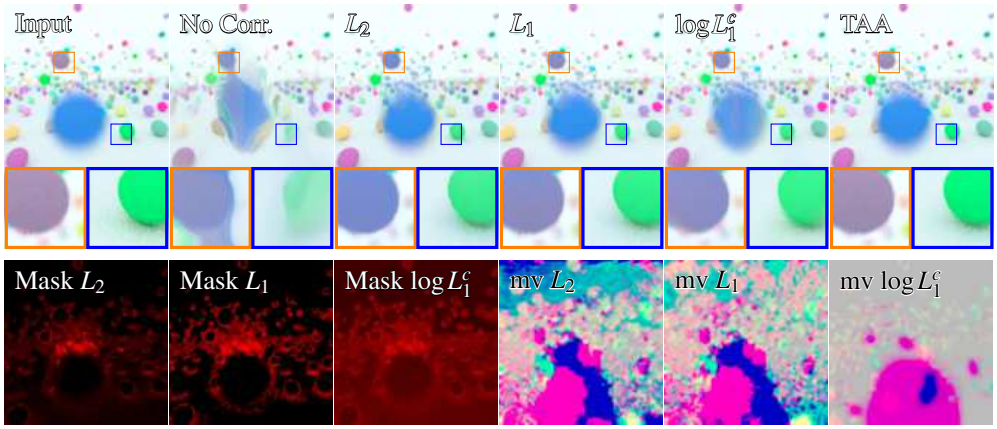

Figure 5. A challenging test case, illustrating the behavior of different error norms: the spheres and camera are moving very quickly here. From left to right: one frame of noisy input, the reprojected and blended frames without any correction applied, using the L_2 error from Equation (2) to reject disocclusions and alignment errors, using the L_1 error from Equation (3), using the $\log L_1^c$ error from Equation (4), and using the standard TAA box rejection method. The second row shows that the error masks are quite similar. We used the mask on the first downsampled level, because the finest level is noisy in this case. Note that κ and η are adjusted according to the norm. The last three images in this row show motion vectors. The cleanest results are obtained using the regularized $\log L_1^c$ error from Equation (5).

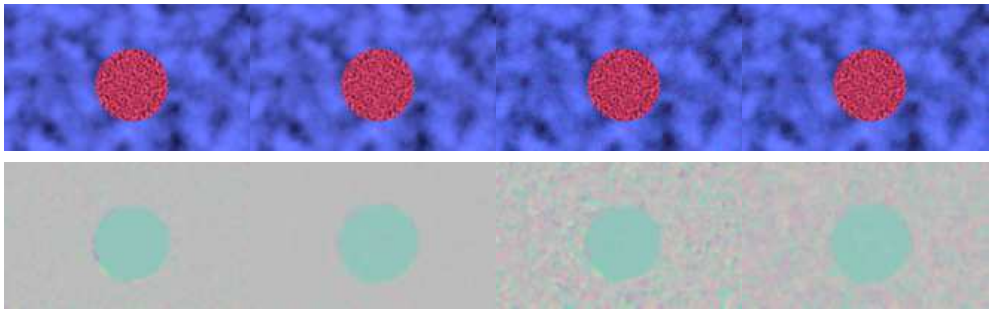

Figure 6. Testing noise resilience of the proposed dense correspondence algorithm. The synthetic disc is moving from left to right. We added white Gaussian noise with a magnitude of 2% and 10% of the clipping threshold, and we tested small blur radii (2, 2, 2, 2) and large radii (32, 16, 8, 8). From left to right: 2% noise, small blur; 2% noise, large blur; 10% noise, small blur; and 10% noise, large blur.

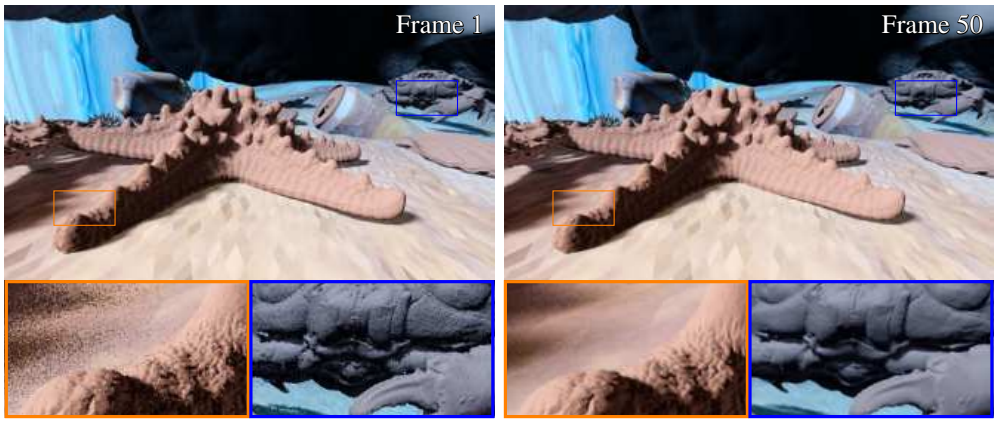

Figure 7. Temporal antialiasing in a game-style setting (rendered at 2048×1152). The geometry has 22M triangles, some of them smaller than a pixel on screen, causing geometric aliasing. On an RTX 2080Ti the full-resolution alignment took 5.44 ms, and the half resolution alignment took 1.96 ms. Please also see the supplemental videos for camera motion with mouse and keyboard interaction.

4. Applications

We implemented the proposed hierarchical alignment method in Vulkan/GLSL compute shaders and tested its performance on three applications, described in the following: temporal antialiasing, aligning photographs, and Monte Carlo rendering.

Temporal antialiasing. Here, we blend $(1-\alpha)$ times the new frame with a fraction of α times the warped old frame. This results in an exponential averaging scheme. We use the regularized $\log L_1^c$ distance and apply the established variance-clipping technique [Yang et al. 2020]: the mean and variance of the new render is estimated in a 3×3 neighborhood, and the color of the warped buffer is clamped to a box with radius $\sigma = 1$ for every color channel. Using this safeguard, we do not need the error mask computed from the patch-based distance to reject alignment errors. Instead, we use it to detect disocclusions and regions of change. In TAA, these are the regions where averaging should take place, because in this application the image is relatively noise free in flat regions but needs smoothing at the occluding geometric edges. We thus invert the mask and apply the averaging only where needed. A result can be seen in Figure 1 (top), which has slow forward camera movement, at subpixel speeds. Without the subpixel motion refinement step, the TAA video shows noticeable stuttering. The scene also features ambiguous, non-rigid motion in the clouds.

Figure 7 shows a version using subpixel jittered ray tracing for image formation along with Monte Carlo integration for soft shadows. These images use variance clipping and Catmull–Rom resampling of the warped buffer. The motion vectors are estimated from relatively noise-free channels (diffuse albedo and normal, only ex-

hibiting geometric aliasing but no Monte Carlo noise). The high geometric complexity along with low-noise features make the motion estimation more robust. However, the geometric aliasing, caused by tiny triangles disappearing between the pixel grid, can cause some pulsing at certain distances from the camera. In the supplemental material, we provide a video with fast user interaction.

This application uses standard variance clipping as a rejection scheme to avoid ghosting artifacts, though this is not ideal to reduce Monte Carlo noise. In the supplemental video, we thus use variance clipping with an adaptive threshold: the tolerance is decreased if the auxiliary buffers show significant differences, because we interpret such differences as misalignment or geometric aliasing. On the other hand, the shadows will clip variance at a rather loose 2σ bound.

We use arbitrary output variables for motion detection here, because these are often available. Likewise, if true motion vectors are available, these can be used at any scale and for any masked region in the image to guide or overwrite our matching procedure for more stable results.

Aligning photographs. We used our alignment procedure to drive a denoising application based on burst photography, similar to a mobile camera technique [Hasinoff et al. 2016]. This application depends on fast computation for non-destructive workflows on desktop machines, where user interaction can trigger a complete re-evaluation of the processing graph. Moreover, it has a crucial performance constraint if deployed on embedded devices such as camera firmware or as smartphone software. For this, we merged six raw shots of a 16MP Fujifilm camera before demosaicing. Due to the X-Trans sensor layout (which can be analysed in 3×3 blocks), the aligned buffers have 1632×1088 pixel resolution, and the subpixel refinement lifts it to the full resolution of 4896×3264 after demosaicing. We use the L_1 distance here. The full pipeline takes 33.9 ms, out of which 19.4 ms are spent aligning the six images. On loading, 6×2.5 ms are needed to upload the images to the GPU. The input images and the result can be seen in Figures 8 and 9. When blending the buffers, we use a Gaussian/Poissonian noise model that was fitted to the camera at this ISO beforehand [Foi et al. 2008].

Monte Carlo rendering. In the DINING TABLE scene (Figure 10) we tested a very noisy diffuse global illumination render. We use a large blur radius of 10 pixels for the finest scale, 4 on the next-finer scale, and the default 2 otherwise. The figure shows split-screen renders with the 1 spp input on the left and our aligned/merged version on the right. Even in the presence of this amount of noise, we can track features and align the images. Total frame times were 5.7 ms, out of which 3.1 ms were spent on ray tracing and 2.4 ms on aligning at full resolution (1024×1024). The split screen is applied after rendering the full image.

The SUBSURFACE SCATTERING scene (Figure 11) shows the searchlight prob-

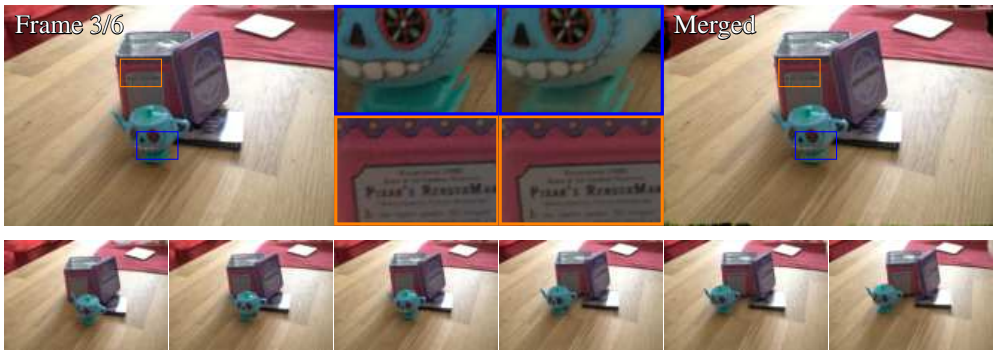

Figure 8. Moving teapot. Bottom: A sequence of six handheld photographs taken in burst mode, at ISO 3200 and pushed by +2.5ev in post. We can align these images despite the walking teapot moving and perform a noise-based merging. Processing takes 34 ms. No denoising other than averaging the aligned images has been performed.

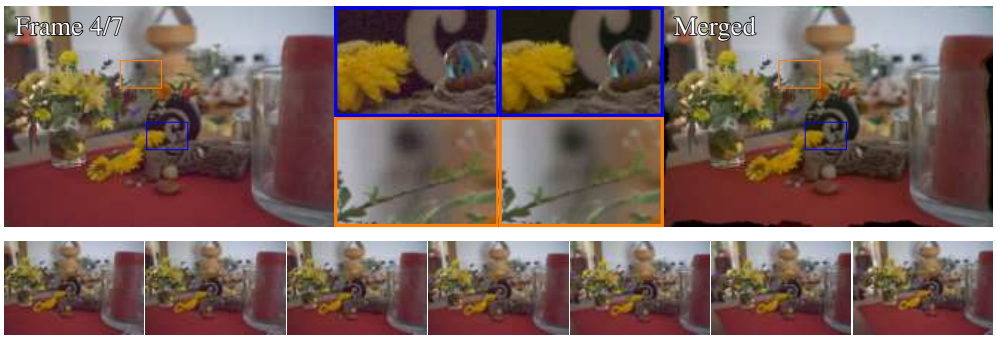

Figure 9. Still life. Bottom: A sequence of six handheld photographs taken in burst mode, at ISO 3200 and pushed by +3.5ev in post. We can align these seven images and perform noise-based merging. Processing takes 30 ms, out of which 22 ms are aligning and merging. There is light wavelet denoising applied to the result. See the supplemental material for a comparison with and without additional denoising.

lem: a slab of path-traced subsurface scattering with a highly anisotropic phase function (mean cosine 0.97). Over the course of the animation, the incident light beam is tilted from normal incidence ($\theta = 0$) to the grazing angle ($\theta = \pi/2$). The appearing scattering pattern changes in non-rigid ways, and our alignment helps moving noise patterns from the previous frames. This leads to good noise reduction, making it possible to judge the resulting surface brightness and color from the preview render. For low sample counts and high variance, this is a problem because of the nonlinear output device transform (ODT) [Academy Color Encoding System (ACES) Project Committee 2014]: besides applying the tone reproduction curve, it clips outlier samples to the maximum display value and thus leaves less energy in the image than noise-free samples would. Hence, previews of noisy renders often look much

Figure 10. A few frames from the DINING TABLE scene, rendered with three bounces of global illumination at one sample per pixel, all diffuse and with motion blur. The small insets show the initial frames of the sequence (every fourth frame is shown), where the system still needs to adjust. The larger images show frames 180 and 250. Every image has a split screen: the 1 spp input (left) and the aligned and merged output (right).

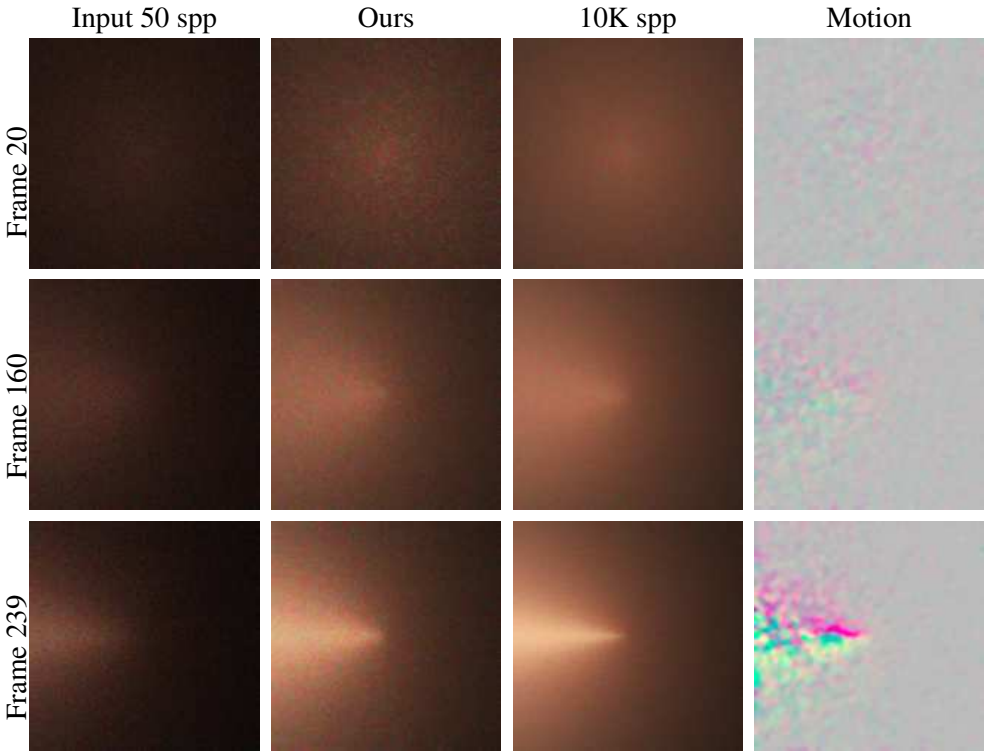

Figure 11. Subsurface scattering: frames 20, 160, and 239 from an animation. The angle of incidence of a light beam illumination on the surface changes from orthogonal (frame 0: $\theta = 0$) to grazing (frame 240: $\theta = \pi/2$). Our alignment and merging (second column) helps moving the bright areas in previous frames to match the shape of the scattering lobe in the new frame, making it possible to predict the brightness and color of the final render (third column) much better than the input at 50 spp (first column).

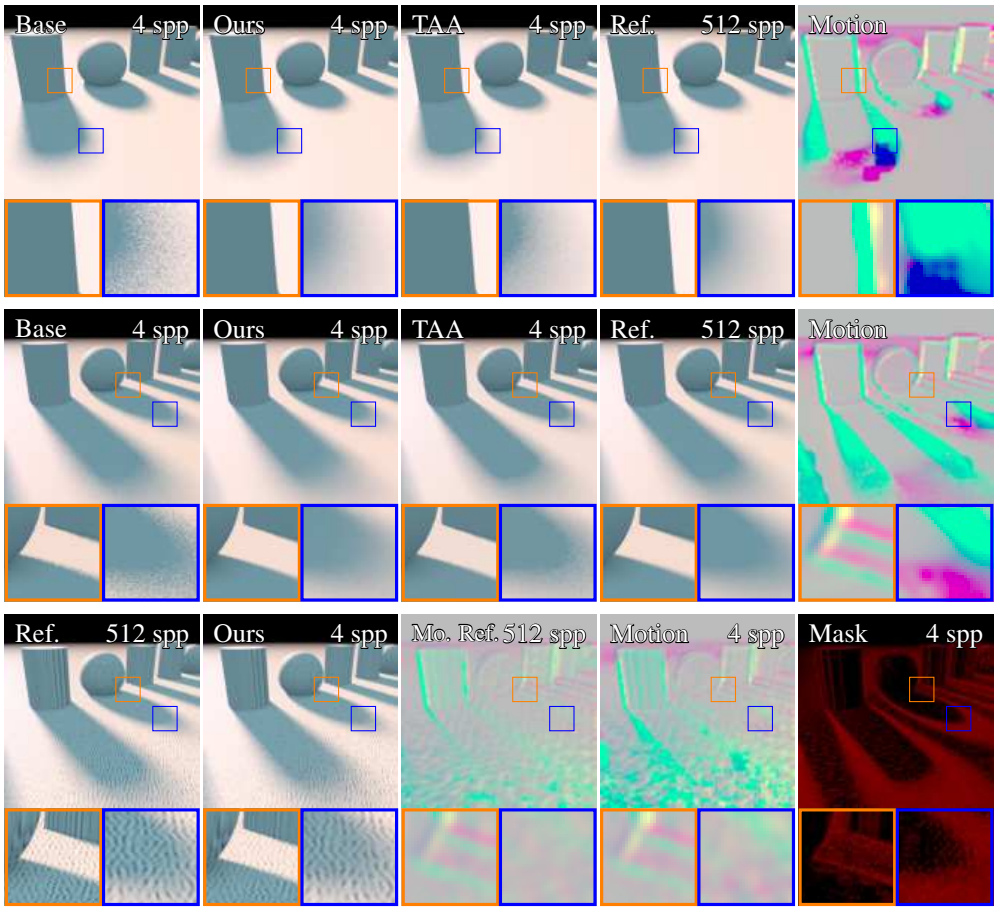

Figure 12. Frames 38 (top row) and 102 (middle row) from an animation with moving light source and camera. This causes the objects in world space to have different motion than the shadow boundaries. At only 4 spp our method manages to both antialias edges (orange insets) and show shadows (blue insets) close to the much more expensive 512 spp reference. Note that TAA box rejection creates unpleasant quantization artifacts in this case. The bottom row shows our motion vectors in the case that the surface texture is not demodulated. Now, we need to track two contradicting motions at the same time, which results in slightly degraded noise removal performance due to the error mask that detects contradicting motion.

too dark. This application uses the $\log L_1^c$ norm as the patch distance, because it is more resilient to the outlier noise often encountered in Monte Carlo estimates than the L_2 norm. We do not use any additional variance-clipping rejection, in contrast to the TAA application.

The PENUMBRA scene (Figure 12) shows the incident illumination on a scene with moving camera and moving light, featuring soft shadows. The motion is ambiguous, because the shadow and the geometry move in different ways. Such a buffer

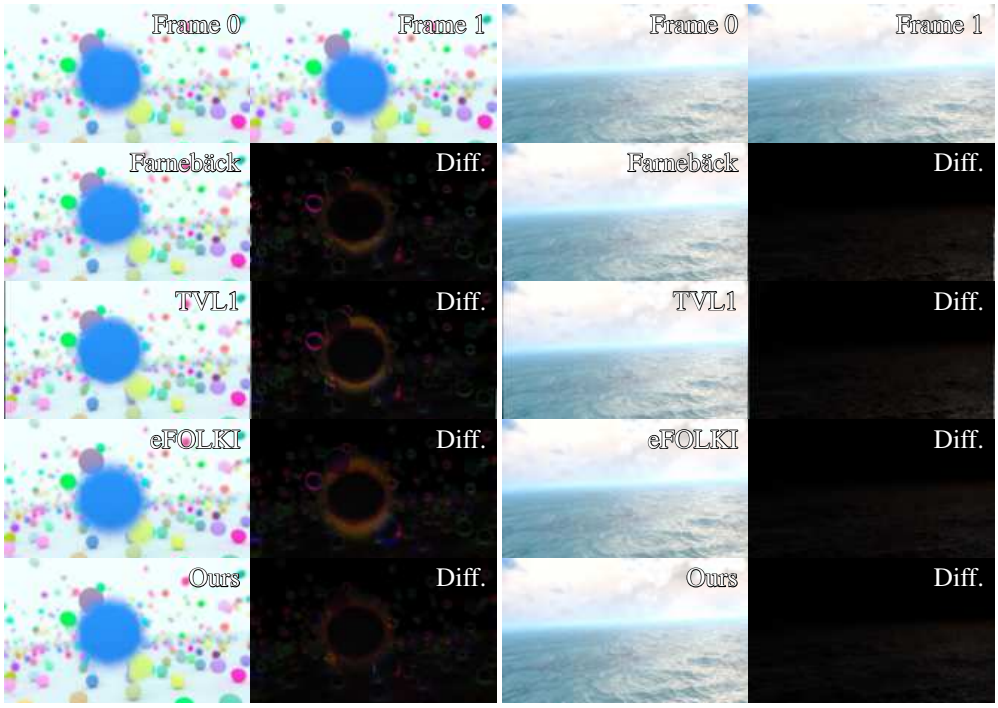

Figure 13. Comparison to optical flow methods. From top to bottom: input frames, Farnebäck, DUAL-TVL1, eFOLKI, and ours. The *diff.* images show the absolute difference, where more black and darker pixels are better.

is often obtained by albedo demodulation, which is a common practice in real-time denoising [Chaitanya et al. 2017]. Our alignment correctly registers soft shadows and sharp edges and results in antialiased shapes as well as smooth shadows. However possible, it would be involved in deriving motion vectors for soft shadow boundaries, and such special-case algorithms would need to be implemented for every path-space effect separately. To result in cleaner motion vectors, we used the regularized $\log L_1^c$ distance here.

5. Evaluation

Comparison to optical flow. Figures 13 and 14 show two consecutive frames of an animation, as well as the warped versions when computing the dense pixel correspondences with various techniques. We compared two well-established optical flow methods (Farnebäck [Farnebäck 2003] and TVL1 [Sánchez Pérez et al. 2013]) because their implementations are readily available in OpenCV. We also compared against eFOLKI [Plyer et al. 2016] as a close contender because of its speed (see [Butler et al. 2012] for a performance comparison).

We show the absolute difference between the warped frames and the ground truth.

Figure 14. Comparison to optical flow methods on two sequences from the Sintel dataset. From top to bottom: input frames, Farnebäck (0.100 s), DUAL-TVL1 (12–260 s), eFOLKI (runtime in Python: 7 s), and ours (0.0012 s). The *diff.* images show the absolute difference, where more black and darker pixels are better.

These are relevant to our application, because large pixel-color differences mean that the intra-frame averaging has to reject the aligned pixels. That is, more black pixels and ones with smaller absolute difference values are better. Note that eFOLKI instead minimizes distortion, which may be the better way to fail in different applications.

The SPHERES scene (Figure 13, left) is challenging because it introduces noise from various sources: depth of field, motion blur, and soft shadows. It also has a lot of disocclusion and contradicting incoherent motion, as for instance smaller spheres bouncing in the background of a lens or motion-blurred foreground. For our goal to use optical flow for real-time noise reduction, this case is very important. Our method shows a much-improved difference image as compared to the other three. Note that this is battle-testing our approach because the colorful spheres can best be discerned by their color, while our matching is performed on scalar pixel differences.

The OCEAN scene (Figure 13, right) tests contradicting non-rigid motion, as caused by small wavelets passing over larger waves at a different velocity, as well as specular reflections moving separately. We observe that eFOLKI and our method both work more robustly on these test scenes than the other two methods.

We also provide a test on the Max Planck Institute SINTEL scenes (Figure 14). Our algorithm is more well suited for merging frames, as can be seen because more pixels are darker than for the others and because it is much faster to compute. Note that we are not trying to compute actual flow and that our method is thus not applicable to cases where the real optical flow is needed.

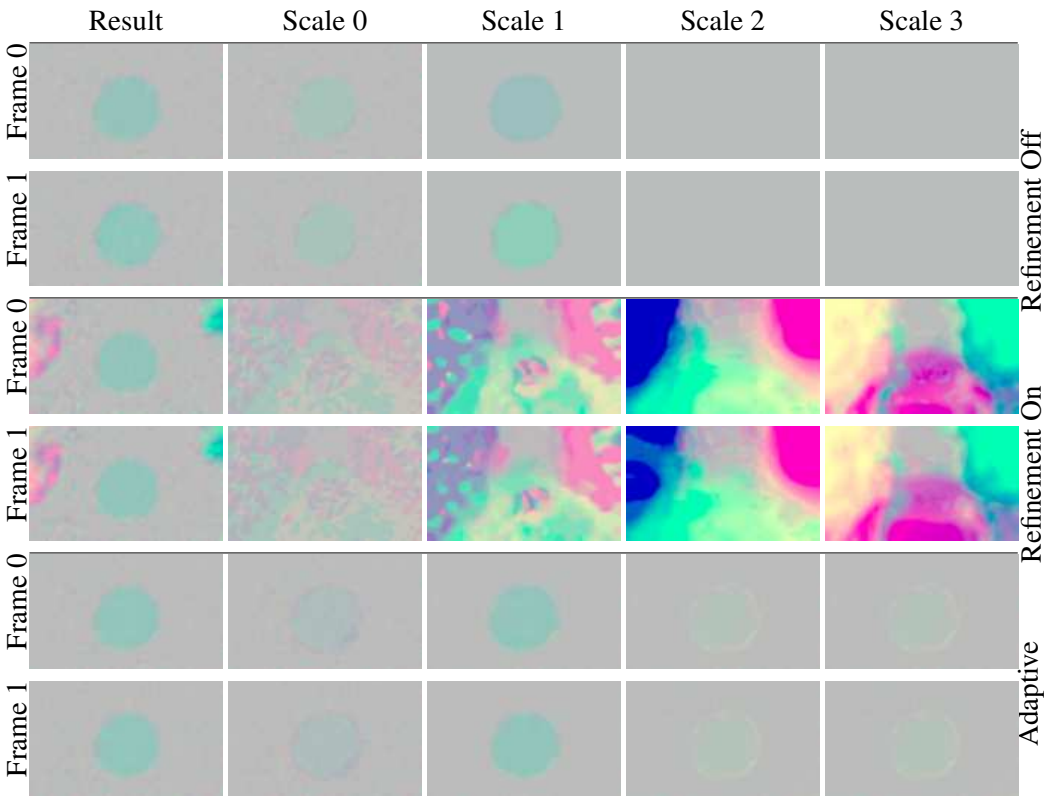

Figure 15. Testing subpixel refinement on different scales, with 2% added noise. The disc here moves slowly, so subpixel velocity jitter is important. The images are grouped into rows for frame 0 and frame 1. Ideally there would be very little difference in motion between frames. Top: Without subpixel refinement. Center: Always refined on all levels. Bottom: Adaptive, i.e., only refined on scales 0 and 1. The images for each scale show the resulting motion vectors when only displacement on the given scale plus potentially subpixel refinement on other scales is considered.

Subpixel motion refinement. In Figure 15 we evaluate the importance of subpixel motion vector refinement on all scales. We evaluate the final outcome as well as the temporal aliasing between two frames. It can be seen that frame 0 and frame 1 show substantially different motion on scale 1 when the subpixel refinement is off, even though the disc moves at a constant speed. We conclude that subpixel refinement is essential to avoid temporal jittering in the motion. This is most apparent in frames 0/1 for scale 1 without refinement, but also carries over to more subtle jitter on the finest scale. On coarse scales, however, there is too much spatial aliasing to compute useful subpixel refinement. Thus, we adaptively switch on the refinement only for the finer half of the scales.

Note that subpixel refinement can fail if the input is extremely noisy, because the

Pass	Dist.	Blur	Merge	Sum
Level 3	0.007	0.003	0.010	0.020
Level 2	0.015	0.006	0.010	0.031
Level 1	0.131	0.031	0.055	0.217
Level 0	2.202	0.361	0.807	3.370
	Down	Warp	Blend	
Overhead	0.140	0.119	0.112	0.371
				4.008

Table 1. Breakdown of our timings for the HIMALAYA scene in Figure 1 at 2350×1000 resolution run on an NVIDIA RTX 2080 Ti. All numbers are in milliseconds. The *down* overhead collects all timings required to compute the four pyramid levels of the two input images. Rendering the scene takes 4.9 ms on this device. The kernels *dist.*, *blur*, and *merge* refer to the kernels introduced in Section 3.1 and are listed separately for each level of the pyramid on which they are run, level 0 being the finest and 3 the coarsest.

Pass	Dist.	Blur	Merge	Sum
Scale 3	0.006	0.004	0.009	0.019
Scale 2	0.010	0.004	0.009	0.023
Scale 1	0.039	0.010	0.019	0.068
Scale 0	0.498	0.100	0.207	0.805
	Down	Warp	Blend	
Overhead	0.118	0.111	0.107	0.336
				1.251

Table 2. Breakdown of our timings for the HIMALAYA scene in Figure 1, in the same setup as Table 1 only using 2×2 downsampling for the motion vector estimation. As expected, mainly the higher resolutions profit from initial downsampling.

noise destroys the underlying smoothness assumption. This means that the quadric fit will not be a meaningful interpolation. Please also see the supplemental material for a comparison video with and without subpixel refinement. It also contains detailed information about memory usage and how it is reduced when working with downscaled buffers for motion detection.

Performance. We show a timing breakdown in Table 1. As expected, most of the time is spent on the high-resolution scale (85–93%). To further speed up the method, it is possible to rely on the subpixel refinement and only perform the alignment up to half resolution. We explored this possibility in an experiment where alignment is performed on a 2×2 downsampled input buffer. For the HIMALAYA scene, runtimes went down from 4.0 ms to 1.25 ms, with only minor differences in output quality (see

Figure 16. Frame 200 of the HIMALAYA sequence when computing the full-resolution correspondences (235 × 1000 px, 4.00 ms, left), or based on a 2 × 2 subsampled version (1.25 ms, center). The absolute difference (right) is 8×.

Table 2 and Figure 16). This is also exploited in the photography application, where 3 × 3 downsampling is done (Figure 8).

We qualitatively compared our method to the result of the eFOLKI method [Plyer et al. 2016] by running the CPU/Python code provided by the authors (`https://github.com/aplyer/gefolki`). To gauge the relative performance to their GPU implementation, we can approximately normalise the runtimes for compute power or bandwidth of the utilised devices. We arrive at roughly a 1.4–1.8× speedup or 4.5–5.0× for the 2×2 downsampled version of our method.

Memory requirements. When running the shader kernels, we use a memory allocator to manage the input and output image buffers. That is, the memory is aliased and can be reused by later kernel invocations in the command buffer. Thus, we provide peak resident set size (rss) as well as address space size (vmsize) numbers when evaluating memory usage. In general, the hierarchical matching requires single-channel versions of a pyramid of the source and target image buffers. The largest resolution is either the input size or the size downsampled by 2 × 2. For instance, a full pipeline for a 1024 × 1024 render takes 17 MB rss of image buffers without any alignment. These base buffers are used for intermediate outputs of the rendering kernel, the blend module, and an A/B split-screen comparison module as well as blue noise input textures. The full-resolution alignment pipeline then takes 71 MB, and the half-resolution alignment pipeline takes 35 MB rss.

Figure 7 shows a more complex scene with millions of triangles and large textures. The memory requirements without any alignment kernels are rss 733.953 MB and vmsize 733.953 MB, at 2048 × 1152 resolution. The complete memory requirements for all geometry and textures and the intermediates for alignment were peak

rss 895.562 MB and vmsize 932.72 MB for full resolution, and peak rss 806.312 MB and vmsize 828.094 MB for half-resolution.

Please see the supplemental material for more numbers.

Limitations and fail cases. The hierarchical approach assumes that there is some coherence of motion. Arbitrary correspondences can only be established within the 5×5 window on every level of the pyramid. Relying on an upsampled version of a coarser-level motion vector to extend the range means that a whole block on the finer level will start the search in a similar region. Even in the presence of incoherent motion, often the results are still a good match to the pixel surroundings, but this can cause some random buzzing in animations, which has to be addressed in a rejection step. Please refer to the supplemental videos of the underwater scene for a comparison of different rejection approaches. In the future we hope that edge-aware upsampling can improve such cases.

Excessive noise in the input will lead to lower accuracy in the output. Applications with a large amount of noise can also not rely on the quadric fit for refinement, because a polynomial fit to noisy values is not meaningful.

We presented an efficient correspondence detection method, but did not explore good averaging/rejection strategies tailored for specific use cases.

6. Conclusion and Future Work

We proposed a hierarchical algorithm for fast alignment of subsequent frames in real-time rendering. This has many use cases also for interactive preview rendering or for faster denoising in offline rendering. Our approach is completely deterministic and non-iterative and, though bearing similarity to neural networks in the data flow diagrams, can be evaluated completely without optimizing weights. It can, however, serve as input to sophisticated denoising methods, some based on neural networks [Chaitanya et al. 2017; Thomas et al. 2020; Xu et al. 2019], as these depend on motion vectors or pre-warped input frames. We showed that, at similar quality, the matching speed can be improved substantially by running on reduced resolution and relying on the subpixel motion vector estimation for the final upsampling step.

There are many things to be explored to improve our work. First, the alignment quality might be improved by exploiting forward and backward searches, if all frames are available (as for offline denoising). The hierarchical scheme might be improved by using more sophisticated bilateral/edge-aware upsampling instead of the simple bilinear scaling we used. The enforced coherence between the hierarchical levels may be broken up to result in more precise alignment by using an iterative refinement as a post-process, if the application allows for more budgeted render time.

We demonstrated fast speed in a large variety of situations: even though standard TAA with motion vectors is still at least one order of magnitude faster for certain

specific scenarios, our solution is more general and can be applied in cases where there are no simple motion vectors. With increasing visual complexity, we expect this to be of importance going forward. More complex light transport will require more sophisticated temporal reuse. We think advanced techniques such as image-space control variates [Rousselle et al. 2016] could also greatly benefit from alignment.

Acknowledgements

The HIMALAYA and SPHERES scenes were created by Reinder Nijhoff:

https://www.shadertoy.com/view/MdGfzh
https://www.shadertoy.com/view/lsX3DH

The PENUMBRA scene were created by Inigo Quilez:

https://www.shadertoy.com/view/WdyXRD

The OCEAN scene was created by Alexander Alekseev:

https://www.shadertoy.com/view/Ms2SD1.

References

ACADEMY COLOR ENCODING SYSTEM (ACES) PROJECT COMMITTEE, 2014. Technical Bulletin TB-2014-004 Informative Notes on SMPTE ST 2065-1 – Academy Color Encoding Specification (ACES). URL: http://j.mp/TB-2014-004. 32

BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH) 28*, 3, 24:1–24:11. URL: https://doi.org/10.1145/1576246.1531330. 23

BEKAERT, P., SBERT, M., AND HALTON, J. 2002. Accelerating path tracing by re-using paths. In *Eurographics Workshop on Rendering*, Eurographics Association, 125–134. URL: https://dx.doi.org/10.2312/EGWR/EGWR02/125-134. 20, 23

BLINN, J. F. 1982. A generalization of algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (July), 235–256. URL: https://doi.org/10.1145/357306.357310. 21

BUADES, A., AND LISANI, J. 2020. Enhancement of noisy and compressed videos by optical flow and non-local denoising. *IEEE Transactions on Circuits and Systems for Video Technology 30*, 7, 1960–1974. URL: https://doi.org/10.1109/TCSVT.2019.2911877. 22

BUADES, A., COLL, B., AND MOREL, J. M. 2005. Denoising image sequences does not require motion estimation. In *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance*, IEEE Computer Society, 70–74. URL: https://doi.org/10.1109/AVSS.2005.1577245. 21, 22

BUADES, A., COLL, B., AND MOREL, J.-M. 2005. A non-local algorithm for image denoising. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, 60–65. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2005.38. 23

BUTLER, D. J., WULFF, J., STANLEY, G. B., AND BLACK, M. J. 2012. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision (ECCV)*, Springer-Verlag, no. 7577 in Lecture Notes in Computer Science, 611–625. URL: https://doi.org/10.1007/978-3-642-33783-3_44. 22, 35

CHAITANYA, C. R. A., KAPLANYAN, A. S., SCHIED, C., SALVI, M., LEFOHN, A., NOWROUZEZAHRAI, D., AND AILA, T. 2017. Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics 36*, 4 (July), 98:1–98:12. URL: https://doi.org/10.1145/3072959.3073601. 35, 40

DARBON, J., CUNHA, A., CHAN, T., OSHER, S., AND JENSEN, G. 2008. Fast nonlocal filtering applied to electron cryomicroscopy. In *Proceedings of the IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, IEEE Computer Society, 1331–1334. URL: https://doi.org/10.1109/ISBI.2008.4541250. 24, 25

FARNEBÄCK, G. 2003. Two-frame motion estimation based on polynomial expansion. In *Scandinavian Conference on Image Analysis*, Springer-Verlag, no. 2749 in Lecture Notes in Computer Science, 363–370. URL: https://link.springer.com/chapter/10.1007%2F3-540-45103-X_50. 22, 35

FASCIONE, L., HANIKA, J., HECKENBERG, D., KULLA, C., DROSKE, M., AND SCHWARZHAUPT, J. 2019. Path tracing in production: Part 1: Modern path tracing. In *ACM SIGGRAPH 2019 Courses*, Association for Computing Machinery, 19:1–19:113. URL: https://doi.org/10.1145/3305366.3328079. 20, 23

FOI, A., TRIMECHE, M., KATKOVNIK, V., AND EGIAZARIAN, K. 2008. Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data. *IEEE Transactions on Image Processing 17*, 10, 1737–1754. URL: https://doi.org/10.1109/TIP.2008.2001399. 31

HASINOFF, S. W., SHARLET, D., GEISS, R., ADAMS, A., BARRON, J. T., KAINZ, F., CHEN, J., AND LEVOY, M. 2016. Burst photography for high dynamic range and low-light imaging on mobile cameras. *ACM Transactions on Graphics 35*, 6, 192:1–192:12. URL: https://doi.org/10.1145/2980179.2980254. 22, 26, 31

KAWASE, M., 2003. Frame buffer postprocessing effects in *DOUBLE-S.T.E.A.L (Wreckless)*. Game Developer Conference Presentation. URL: http://www.daionet.gr.jp/~masa/archives/GDC2003_DSTEAL.ppt. 26

LE BESNERAIS, G., AND CHAMPAGNAT, F. 2005. Dense optical flow by iterative local window registration. In *IEEE International Conference on Image Processing*, IEEE Computer Society, vol. 1, I–137. URL: https://doi.org/10.1109/ICIP.2005.1529706. 22, 28

LIBA, O., MURTHY, K., TSAI, Y.-T., BROOKS, T., XUE, T., KARNAD, N., HE, Q., BARRON, J. T., SHARLET, D., GEISS, R., ET AL. 2019. Handheld mobile photography in very low light. *ACM Transactions on Graphics 38*, 6 (Nov.), 164:1–164:16. URL: https://doi.org/10.1145/3355089.3356508. 22

NEHAB, D., SANDER, P. V., LAWRENCE, J., TATARCHUK, N., AND ISIDORO, J. R. 2007. Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, Eurographics Association, 25–35. URL: http://gfx.cs.princeton.edu/pubs/Nehab_2007_ARS/NehEtAl07.pdf. 23

OLEJNIK, M., AND KOZŁOWSKI, P., 2020. Raytraced shadows in Call of Duty: Modern Warfare. URL: https://www.activision.com/cdn/research/Raytraced_Shadows_in_Call_of_Duty_Modern_Warfare.pdf. 23

PHARR, M., JAKOB, W., AND HUMPHREYS, G. 2016. *Physically Based Rendering: From Theory to Implementation*, 3rd ed. Morgan Kaufmann Publishers Inc. URL: https://www.pbrt.org. 20

PLYER, A., BESNERAIS, G., AND CHAMPAGNAT, F. 2016. Massively parallel Lucas Kanade optical flow for real-time video processing applications. *Journal of Real-Time Image Processing volume 11*, 4 (Apr.), 713–730. URL: https://doi.org/10.1007/s11554-014-0423-0. 22, 35, 39

ROEGNER, C., WIMMER, M., HANIKA, J., AND DACHSBACHER, C. 2015. Image-based reprojection using a non-local means algorithm. Tech. Rep. TR-186-2-05-2, Institute of Computer Graphics and Algorithms, Vienna University of Technology. URL: https://jo.dreggn.org/home/2015_reproject.pdf. 23

ROUSSELLE, F., JAROSZ, W., AND NOVÁK, J. 2016. Image-space control variates for rendering. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia) 35*, 6, 169:1–169:12. URL: https://doi.org/10/f9cphw. 41

SÁNCHEZ PÉREZ, J., MEINHARDT-LLOPIS, E., AND FACCIOLO, G. 2013. TV-L1 Optical Flow Estimation. *Image Processing On Line 3*, 137–150. URL: https://doi.org/10.5201/ipol.2013.26. 22, 35

SCHERZER, D., JESCHKE, S., AND WIMMER, M. 2007. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, Eurographics Association, 45–50. URL: https://dl.acm.org/doi/10.5555/2383847.2383856. 23

SCHIED, C., PETERS, C., AND DACHSBACHER, C. 2018. Gradient estimation for real-time adaptive temporal filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2, 24:1–24:16. URL: https://doi.org/10.1145/3233301. 21, 23

STAM, J., AND SCHMIDT, R. 2011. On the velocity of an implicit surface. *ACM Transactions on Graphics 30*, 3, 21:1–21:7. URL: https://doi.org/10.1145/1966394.1966400. 21

TATARCHUK, N., HILLAIRE, S., STACHOWIAK, T., BOWLES, H., WANG, B., SILVENNOINEN, A., AND TIMONEN, V. 2015. Advances in Real Time Rendering, Part I. In *ACM SIGGRAPH 2015 Courses*. URL: https://doi.org/10.1145/2776880.2787701. 21

THOMAS, M. M., VAIDYANATHAN, K., LIKTOR, G., AND FORBES, A. G. 2020. A reduced-precision network for image reconstruction. *ACM Transactions on Graphics 39*, 6 (Nov.), 231:1–231:12. URL: https://doi.org/10.1145/3414685.3417786. 40

WRONSKI, B., GARCIA-DORADO, I., ERNST, M., KELLY, D., KRAININ, M., LIANG, C.-K., LEVOY, M., AND MILANFAR, P. 2019. Handheld multi-frame super-resolution. *ACM Transactions on Graphics* (July), 28:1–28:18. URL: https://doi.org/10.1145/3306346.3323024. 22

XU, B., ZHANG, J., WANG, R., XU, K., YANG, Y.-L., LI, C., AND TANG, R. 2019. Adversarial Monte Carlo denoising with conditioned auxiliary feature modulation. *ACM Transactions on Graphics 38*, 6 (Nov.), 224:1–224:12. URL: https://doi.org/10.1145/3355089.3356547. 40

YANG, L., LIU, S., AND SALVI, M. 2020. A survey of temporal antialiasing techniques. *Computer Graphics Forum 39*, 2. URL: https://doi.org/10.1111/cgf.14018. 23, 30

ZIMMER, H., ROUSSELLE, F., JAKOB, W., WANG, O., ADLER, D., JAROSZ, W., SORKINE-HORNUNG, O., AND SORKINE-HORNUNG, A. 2015. Path-space motion estimation and decomposition for robust animation filtering. *Computer Graphics Forum 34*, 4, 131–142. URL: https://doi.org/10.1111/cgf.12685. 23

ZWICKER, M., JAROSZ, W., LEHTINEN, J., MOON, B., RAMAMOORTHI, R., ROUSSELLE, F., SEN, P., SOLER, C., AND YOON, S.-E. 2015. Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. *Computer Graphics Forum (State of the Art Reports) 34*, 2, 667–681. URL: https://doi.org/10.1111/cgf.12592. 20

Index of Supplemental Materials

- Supplemental webpage with additional image and video comparisons: https://jo.dreggn.org/supplemental/readme.html
- Source code: https://github.com/hanatos/vkdt/tree/submit/src/pipe/modules/burst

Author Contact Information

Johannes Hanika
Karlsruhe Institute of Technology
Am Fasanengarten 5
76131 Karlsruhe
hanika@kit.edu

Lorenzo Tessari
Karlsruhe Institute of Technology
Am Fasanengarten 5
76131 Karlsruhe
lorenzo.tessari@kit.edu

Carsten Dachsbacher
Karlsruhe Institute of Technology
Am Fasanengarten 5
76131 Karlsruhe
dachsbacher@kit.edu

Johannes Hanika, Lorenzo Tessari, and Carsten Dachsbacher, Fast Temporal Reprojection without Motion Vectors, *Journal of Computer Graphics Techniques (JCGT)*, vol. 10, no. 3, 19–45, 2021
http://jcgt.org/published/0010/03/02/

Received:	2020-11-16		
Recommended:	2021-04-08	Corresponding Editor:	Natasha Tatarchuk
Published:	2021-09-30	Editor-in-Chief:	Marc Olano

© 2021 Johannes Hanika, Lorenzo Tessari, and Carsten Dachsbacher (the Authors).
The Authors provide this document (the Work) under the Creative Commons CC BY-ND 3.0 license available online at http://creativecommons.org/licenses/by-nd/3.0/. The Authors further grant permission for reuse of images and text from the first page of the Work, provided that the reuse is for the purpose of promoting and/or summarizing the Work in scholarly venues and that any reuse is accompanied by a scientific citation to the Work.