ESPReSSo: Efficient Slanted PatchMatch for Real-time Spacetime Stereo

Harris Nover Supreeth Achar Dan B Goldman
Google
{nover,supreeth,dgo}@google.com

(a) (b) (c) (d)

Figure 1: ESPReSSo can work on (a) moving scenes and generate (b) dense, high quality depth maps at 1 megapixel at over 60Hz on a desktop GPU. As a byproduct, our algorithm also generates (c) a per-pixel estimate of surface orientation. The accuracy of the depth maps we output can be seen when (d) textured and reprojected from a novel viewpoint.

Abstract

We present ESPReSSo, the first real-time implementation of spacetime stereo, offering improved quality vs. existing real-time systems. ESPReSSo is a local stereo reconstruction algorithm that precomputes subpixel-shifted binary descriptors, then iteratively samples them along slanted disparity plane hypotheses, applying an edge-aware filter for spatial cost aggregation. Plane hypotheses are shared within rectangular tiles, but every pixel selects a different winner from among these candidates, as in PatchMatch Filter [15]. This architecture performs few descriptor computations but many cost aggregations, and we tune our choice of descriptor and filter accordingly: We propose a new 32-bit binary spacetime descriptor breve *combining the benefits of small spatial extent with robustness to scene motion, and the system aggregates costs using the permeability filter [8], a very efficient edge-aware filter. Our prototype outputs 60 depth frames per second on a desktop GPU, using less than 11ms total computation per frame.*

1. Introduction

Real-time active stereo systems such as PrimeSense [1] have opened new classes of input modalities to computing. But the resolution and accuracy of these systems has trailed that of offline stereo systems. Existing commercial systems don't meet the needs of applications requiring more precision, such as geometric acquisition of non-rigid objects for games or entertainment use cases.

Motivated by the need for a high-accuracy real-time system, we seek to adapt methods pioneered for offline use toward real-time performance on modern hardware. Specifically, by projecting varying patterns over time as well as space, the same number of pixels can be sampled with smaller spatial windows [20, 9]. Thus, matching spacetime blocks of pixels can result in higher-quality reconstructions. Although this approach uses active illumination, it falls under the category of "assisted stereo" because it does not rely on any specific calibrated structured light source. This property makes it possible to use multiple such devices simultaneously without the need for special-case handling of illumination interference.

In this work we demonstrate a method for real-time spacetime stereo matching using commodity GPUs. Our method uses a robust 32-bit binary descriptor, and a randomized slanted-window search with block cost filtering, adapted loosely from PatchMatch Filter.

Our method performs matching across short five-frame temporal windows, using dot-pattern illuminators for four infrared frames and ambient IR illumination during a fifth "guide image" used for edge-aware cost filtering. We cap-

ture in the infrared spectrum to avoid interference with a synchronized RGB sensor. Our descriptor is engineered around this five-pair configuration to balance spatial compactness with robustness to motion. The system precomputes frontoparallel descriptors – that is, axis-aligned to the spacetime pixel grid – then tests regional plane hypotheses by aggregating and filtering Hamming distances between these precomputed descriptors. Our edge-aware filter is the permeability filter – a fast IIR filter – making it possible to filter and test 384 disparity planes for each output pixel at over 60Hz.

The benefit of using spacetime illumination patterns is higher accuracy, thanks to both spatially compact descriptors and cost aggregation over smaller spatial windows. Descriptors with very small spatial extent (3×3) are highly resistant to edge fattening artifacts, and enable the use of nearest-neighbor lookup on slanted disparity planes, even using a lookup table precomputed on a frontoparallel spacetime grid. Thus, the costs of temporal aggregation and descriptor computation are incurred only once in a precomputation step, and we compute usable depth data even for highly oblique surfaces and near depth discontinuities, where many other real-time methods discard pixels.

Our contributions include a novel refactoring of stereo matching using descriptor precomputation with slanted plane cost aggregation, a spacetime binary descriptor that is robust to both scene motion and highly oblique geometry, and the use of a permeability filter for slanted plane cost filtering. These developments enable what is, to our knowledge, the first demonstration of real-time spacetime stereo.

2. Related work

Spacetime stereo [19, 20, 9] utilizes time-varying illumination patterns to perform stereo matching robustly with small aggregation windows. In particular, Zhang *et al*. [20] showed that shearing spacetime windows can model both tilted planes and deformations over time. In this work we use time-varying illumination patterns and tilted disparity planes, but we do not explicitly model deformation over time, because our time windows are much shorter.

Many stereo methods assume that disparity is locally constant within a small spatial window, but this assumption is violated at depth discontinuities. Better robustness to these edges can be achieved with adaptive support weights, for example via the bilateral filter [17], and more recently the guided filter [13]. Hosni *et al*. demonstrated a GPU implementation of this method, albeit with a large constant. Our work uses the permeability filter [8], a recursive filter specifically designed for stereo matching. Chang *et al*. [7] implemented permeability filtering on an FPGA, illuminating the scene alternately with dot-pattern and flood IR patterns, using the patterns for matching and flood for edge guidance. We employ a similar scheme, but using four different patterns for robust spacetime matching.

Even with adaptive supports, stereo methods using fronto-parallel window matching can perform poorly on oblique surfaces. Bleyer *et al*. [5] implemented stereo matching using tilted disparity planes with bilateral filtering, utilizing the PatchMatch sampling strategy to handle the increased degrees of freedom in the search space. Subsequent variations of PatchMatch stereo have been implemented on the GPU [21, 2]. Further improvements have employed global optimization, including belief propagation [4], variational optimization for a single view [12] and a multiview variational approach [11].

Local approaches can be accelerated by testing planes against super-pixel regions rather than individual pixels, and then filtering costs across those regions using a constant-time edge-aware filter [15]. This approach has been extended to speed up PatchMatch belief propagation [14]. We adopt some aspects of this framework, but our method uses regularly tiled regions, rather than superpixels. The local expansion method proposed by Taniai *et al*. [16] also operates over regular tiles, but does not lend itself easily to real-time implementation.

Nonlinear descriptors such as the census transform [18] and randomly-generated variants [6] can improve match robustness to noise and image brightness variation. More recently, HashMatch [10] applied machine learning to design highly discriminative descriptors.

3. System design

Our algorithm processes images in groups of ten: five synchronized pairs from a reference and secondary camera[1]. During the first two and last two exposures, the scene is illuminated using projected dot patterns, while the middle "guide image" uses only flood illumination.

Each group of ten images is processed as follows: First, in a preprocess, compact spacetime *breve* descriptors are computed for both reference and secondary image stacks. For improved accuracy, we also optionally shift the secondary camera images by subpixel offsets, and compute *breve* descriptors on those shifted images. (The secondary guide image is presently unused.)

The images are then split into independent output tiles of size 32×28, with overlapping input regions of size 36×32. These tiles are then processed in parallel, similarly to the irregular regions of the PatchMatch Filter approach: The system iterates between testing a number of plane hypotheses for each tile, then generating new plane hypotheses using a PatchMatch-style sampling strategy [5], including both propagation from neighboring tiles and random sampling around the current best depths.

[1] We refer to *reference* and *secondary* cameras rather than left and right. Each output depth pixel corresponds to an input pixel in the rectified reference image.

The plane testing loop itself has four sub-steps. First, the plane hypothesis provides a disparity per pixel, which is used to resample the nearest neighbor secondary descriptors. (Aliasing is reduced by using descriptors computed at subpixel offsets, as described above.) Second, raw costs are computed using Hamming distance between descriptors. Third, the raw costs are then filtered using an inexpensive edge-aware permeability filter, to compensate for regions with low signal. Finally, depths are updated for any pixels with lower cost for this plane than the previous best cost. Then, in a postprocess, we compute a mask indicating pixels likely to have invalid depth estimates.

The outer loops of the algorithm described above are summarized in pseudocode in Algorithm 1.

Algorithm 1 Given input image stacks I_{ref}, I_{sec}, ESPReSSo computes a depth map D and valid mask M.

procedure ESPRESSO(I_{ref}, I_{sec})
  $\tilde{I}_{ref} \leftarrow$ RECTIFY(I_{ref})
  $B_{ref} \leftarrow$ COMPUTEDESCRIPTORS($\tilde{I}_{ref}$)
  for subpixel disparities $d \in \{0, \ldots, d_{max}\}$ **do**
    $\tilde{I}^d_{sec} \leftarrow$ RECTIFYWITHXOFFSET(I_{sec}, $\frac{d}{d_{max}+1}$)
    $B^d_{sec} \leftarrow$ COMPUTEDESCRIPTORS($\tilde{I}^d_{sec}$)
  for tile T in reference image domain **do**
    $P \leftarrow$ INITIALIZEPLANES
    $D \leftarrow$ INITIALIZEDEPTHS
    costs $C_{opt} \leftarrow \infty$
  for n iterations **do**
    for tile $T \in B_{ref}$ **parallel do**
      for plane P_i **parallel do**
        $B_{res} \leftarrow$ NNSAMPLE(B_{sec}, P_i)
        $C \leftarrow$ COMPUTECOSTS(B_{ref}, B_{res})
        $C' \leftarrow$ PERMEABILITYFILTER(C)
        $D \leftarrow$ UPDATEDEPTHS(C', C_{opt}, D, P_i)
    if not final iteration **then**
      $P \leftarrow$UPDATEPLANES(D)
  $M \leftarrow$ COMPUTEINVALIDPIXELS(D)
  return D, M

In the next Sections, we will elaborate on the salient components of our system. Section 3.1 introduces our *breve* spacetime descriptor that trades off discriminative power with robustness to motion. In Section 3.2 we describe cost filtering using the permeability filter, and in Section 3.3 we describe how plane hypotheses are initialized and updated. Finally, Section 3.4 discusses our heuristics for invalidating low-confidence depth estimates.

3.1. Spacetime *breve* descriptor

A principal benefit of a spacetime descriptor is that we can shrink the spatial extent to be much smaller than typical single-image descriptors. For example, the BRIEF descriptor [6] uses 9×9 windows, and HashMatch [10] uses 11×11 windows. In contrast, our descriptor is computed over a spacetime window with a spatial extent of only 3×3 pixels and a temporal extent of four exposures. A different random-dot illumination pattern is projected during each exposure. (The fifth image, a guide image with flood illumination, is not used in the descriptor, but will be used for permeability filtering in Section 3.2 below.) As we will see, the spatial compactness of this descriptor provides better robustness to oblique geometry and depth discontinuities.

Let us represent the intensity at rectified pixel (x, y) during pattern t as $\tilde{I}(x, y, t)$. Then each bit k of binary descriptor $\tau(x, y, k)$ compares values of two nearby pixels:

$$\tau(x, y, k) = H(\tilde{I}(x+a_k, y+c_k, t_k) - \tilde{I}(x+b_k, y+d_k, s_k)) \quad (1)$$

where $H()$ denotes the Heaviside step function. The 32-bit binary descriptor B at (x, y) is simply the concatenation of 32 such pair-wise intensity comparisons in the $3 \times 3 \times 4$ spacetime volume centered at pixel (x, y), $B(x, y) = (\tau(x, y, 0), \tau(x, y, 1), \ldots, \tau(x, y, 31))$. This forms a family of descriptors parameterized by various choices of a_k, b_k, c_k, d_k, t_k and s_k. The raw matching cost between two descriptors is simply the Hamming distance, which can be computed very efficiently.

The choice of parameters produces different performance characteristics. For example, one choice of parameters is to adapt the single-image Census [18] and BRIEF [6] descriptors to the spacetime case, by computing them independently for each time-slice and then concatenating the results. Indeed, this initially seems advantageous: Since we do not explicitly account for scene motion during the image capture process, movement can cause high matching costs, especially near depth discontinuities. Using within-image comparisons provides a measure of robustness to motion effects near edge discontinuities.

However, even under high frequency illumination patterns, intensity changes in nearby pixels in the same image are often highly correlated (due to defocus blur, scattering etc). In contrast, our time-varying illumination patterns are uncorrelated across time, so comparisons across time ($t_k \neq s_k$) can improve the discriminative ability of a descriptor. The disadvantage is that comparisons across time are more sensitive to motion than comparisons within the same time-slice.

To combine the benefits of both types of comparisons, we adopted a hybrid scheme that we call *breve*, in which half the comparisons are between randomly selected pixels in the same time slice ($t_k = s_k$) and the rest are between unconstrained random pairs of pixels in the space-time window. (Because the samples are split "half and half," we named the descriptor after the espresso drink *breve*, which uses steamed half-and-half instead of milk.)

Since we are testing slanted planes, it may seem that one would need to resample patches of the secondary images and recompute descriptors for every plane, as part of the plane testing loop. Indeed, an earlier version of our system used this approach. However, much better performance can be obtained by amortizing all image resampling and descriptor computations into a preprocess: We rectify the secondary image with a small number of subpixel translation offsets — using bilinear sampling for rectification — and produce a descriptor for every patch at every offset. (Fusing rectification, subpixel translations, and descriptor computation into one kernel minimizes memory traffic and avoids signal loss due to multiple sampling and reconstruction passes.) The descriptors are then interleaved to create a lookup table of descriptors for every point in disparity space. At plane test time, each pixel's proposed disparity is used to perform a nearest-neighbor lookup into this table.

Surprisingly, this produces results that are as good as or better than the "brute force" approach where descriptors are recomputed for every plane. We hypothesize that our coarse sampling of subpixel disparity space avoids spurious errors in low-signal regions, where highly oblique planes can otherwise match arbitrary patterns in image noise.

3.2. Cost permeability filtering

Having computed the raw costs using the Hamming distance between precomputed descriptors, we aggregate costs intelligently via the edge-aware permeability filter [8], an inexpensive recursive filter briefly summarized here:

Let $C = \mathbb{R}^{w \times h}$ be an image region to be filtered, in our case the raw costs under a given plane hypothesis. The *permeability* $\mu_{P,P'}$ between two adjacent pixels P and P' is a number between 0 and 1 that controls the flow of information between those pixels. A high permeability allows high information flow, and thus larger support windows, while a low permeability allows less information flow, and thus smaller support windows. Permeabilities are symmetric, so that $\mu_{P,P'} = \mu_{P',P}$. We will describe later how permeabilities are computed based on edge strength, but take these as given for the subsequent sections:

We start by applying a simple recursion left-to-right. Let $\mu_{x,y}$ be the permeability between (x, y) and $(x-1, y)$ and define $C^{LR(x,y)}$ as

$$C^{LR}(x, y) = \mu_{x,y} C^{LR}(x-1, y) + C(x, y), \qquad (2)$$

initializing with 0 at the min-x boundary base case. Then we define a right-to-left recursion similarly, noting that because of symmetry we use $\mu_{x+1,y}$ as the permeability between (x, y) and $(x+1, y)$ and

$$C^{RL}(x, y) = \mu_{x+1,y} C^{RL}(x+1, y) + C(x, y), \qquad (3)$$

again using 0 to initialize at the max-x boundary base case. Then the horizontal cost filter $C^H(x, y) = C^{LR}(x, y) + C^{RL}(x, y)$. If we define the permeabilities such that they approach 1 in constant regions and approach 0 at intensity edges, this acts as a one-dimensional edge-aware filter.

The two-dimensional extension of this filter then performs independent top-to-bottom and bottom-to-top passes over C^H, using permeabilities computed between vertical pairs of pixels, producing C^{TB} and C^{BT} respectively. The final cost is similarly defined $C = C^{BT} + C^{TB}$. The resulting weighted support windows for each output pixel mostly track edges, although compared to a more traditional and expensive bilateral or guided filter there are asymmetries. For example, the aggregation region at the center pixel of an hourglass shape is the whole hourglass if the hourglass is vertical but only a line when the hourglass is horizontal.

Computation of this filter is very inexpensive: On architectures such as a GPU where a fused multiply-and-add is the same cost as an add, it requires only 6 operations per pixel. Note that at least 4 operations are required for a recursive box filter, although the permeability filter also requires additional state for intermediate values and permeability coefficients, and the computation is distributed over several passes.

We adopt the same permeability weights as Cigla and Alaton [8], such that

$$\mu^{LR}_{x,y} = e^{\frac{-|G(x,y) - G(x-1,y)|}{\sigma}}, \qquad (4)$$

where σ is a tunable parameter determining the strength of the smoothing, and G is the linear intensity value of the guide image with flood illumination. Although more complicated functions might provide more robustness to image noise, we found this simple approach performs sufficiently well for our application.

Applying this filter on a complete cost volume is straightforward, but what about in our setting, in which we guess a hypothesis for a small tile of pixels? If we only used input pixels within the tile boundaries, points at the boundaries would receive no support from outside the tile, and if their permeabilities on their interior side are also low, they might receive no aggregation at all.

Therefore, we include a small additional apron of pixels around the boundary of the tile in the filter run-up. Increasing the size of this apron improves the quality of the output for edge pixels, but also diminishes efficiency. We've found a 2-pixel apron works well for our purposes, though some artifacts would be reduced with larger aprons. Note that even with aprons, there is asymmetry of costs within a tile: A tile's left edge aggregates costs from many more pixels on the right than the left. However, the permeability values are typically well below 1, which shrinks the effective support distance so that this is not a substantial effect. For efficient GPU implementation, we aggregate over fixed-size tiles independent of scene content, in contrast to the superpixel approach of [15]. In our CUDA implementation, we

wish to align our tiles to maximize the usage for a target warp size of 32 threads, scheduling exactly 32 threads for each pass across a tile. Thus, we compute raw costs for a 36×32 pixel region in the input. The horizontal pass produces intermediate cost buffers C^H over the inner 32×32 pixels, and then the vertical pass produces aggregated costs for the inner 32×28 pixels.

3.3. Plane hypothesis generation

We generate plane hypotheses iteratively using four different heuristics, similar to those in PatchMatch Stereo [5] and PatchMatch Filter [15]:

Initialization: Planes are drawn from a random distribution. Parameterizing each plane as a point and a normal in disparity space, we draw uniform samples over the disparity range, and normals are drawn from a distribution with a frontoparallel bias.

Propagation: For each tile we draw the current best plane sample for a random pixel in its neighboring tiles, in the four cardinal directions.

Perturbation: We pick a random pixel within each tile and perturb its current winning plane in the disparity/normal parameterization described above. The perturbations depend on a scale parameter s, where the amounts to perturb are uniformly distributed between 0 and s. To generate a family of perturbed planes, a series of geometrically decreasing scales $s_j = s_0/\alpha^j$ can be used, effectively sampling from an exponential distribution around the plane.

Hot start: In a real-time system running continuously, a sequence of depth images usually changes relatively slowly over time. We take advantage of this by using the previous output depth image as an estimate for the current frame: For each tile, we test the winning plane from the previous frame at a random pixel in that tile. Although that plane might no longer be a great fit for that particular pixel due to scene motion – especially at moving depth discontinuities – it can still be a reasonable match for other pixels within the same tile.

For efficiency, we tests planes in batches by launching a single CUDA kernel which tests 24 hypotheses per tile, over all tiles at once. So our loop alternates between generating 24 planes to test for each tile, and then testing all those planes together, updating the current winners. We perform 16 iterations per frame, to test a total of 384 planes per tile. In the first iteration we generate all 24 planes via initialization. In the second iteration all 24 planes are generated via hot start. (For the first output image in a sequence, where no previous frame is available, the second iteration randomly initializes 24 more planes.) The next 4 iterations

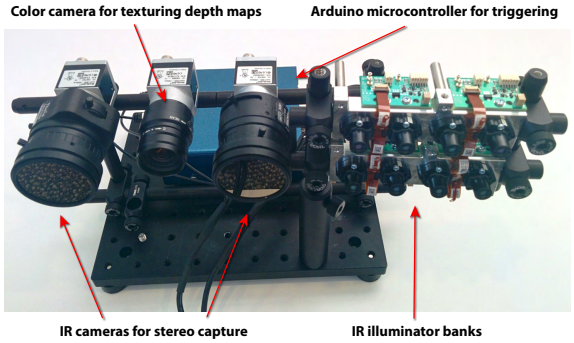

Figure 2: Our capture system uses a pair of cameras (baseline 12cm) with NIR filters. Two banks of four DOEs generate illumination patterns. A color camera produces texture maps. (NIR flood light not shown.)

each consist of 4 planes picked via propagation – one from each cardinal direction – and 20 with perturbation. Here we start at scale $s_0 = 1$ and pick 2 planes at each scale, reducing the scale geometrically by $\alpha = 2^{\frac{8}{10}}$. At this point we assume some winning planes are near the cost minimum. The remaining 10 iterations also choose 4 planes via propagation and 20 for perturbation, but when perturbing planes we start at scale $s_0 = \frac{1}{8}$ in order to refine current winners, rather exploring the full space of planes.

3.4. Depth invalidation

The algorithm described thus far determines a depth estimate for every pixel. However, some points may be occluded from the secondary camera, some surfaces are not reflective enough to be matched, and other regions have transparent or subpixel-width objects in them. As a proxy for detecting these properties, we perform three quick tests to invalidate bad depth estimates: First, we reject any pixels with depth estimates outside the frustum of the secondary camera. Second, we discard pixels with very highly oblique plane tilts; such planes can only have few pixels supporting them and are likely to be outliers. Finally, we determine connected components over the full depth image based on a minimum disparity difference, and discard entire components below a threshold number of pixels. Although these heuristics are not guaranteed to invalidate all bad depth estimates, they are inexpensive and effective in practice.

4. Experiments

For imaging, we use a pair of Basler ace acA1300-200um CMOS cameras fitted with Theia ML410M lenses and near-infrared (NIR) pass filters (Midwest Optics BP850-46). Two banks of NIR diffractive optical elements (DOEs) project time-varying active illumination patterns. Each bank consists of four DOEs, and each patterned image is illuminated by one DOE from each bank. Triggering

two DOEs at a time increases the density of the projected dot pattern, which improves depth accuracy and coverage. During the guide image exposure, the DOEs are turned off and an image is captured with ambient NIR illumination, provided by an untriggered NIR floodlight. The cameras and DOEs are triggered by an Arduino microcontroller. A photo of our prototype system is shown in Figure 2.

As shown in Figure 3, stereo images are captured at 180Hz in a repeating pattern-pattern-guide sequence. Each stereo depth map is computed over a rolling window of 5 frames (4 patterned, 1 guide) centered on a guide image. An RGB camera – used for color texturing – is triggered synchronously with the guide image. This ensures good alignment of depth edges with the associated texture map.

To study the effect of various parameter and algorithmic choices, we cannot use existing datasets, because no ground truth data exists with our space-time capture pattern. Therefore we created our own test sequences moving an object with known geometry in front of our capture system: for the experiments described here, a 3D printed mannequin head[2]. We then use the iterative closest point algorithm [3] to compute the rigid transform that best aligns the true shape with each estimated stereo depth map. We compute statistics on the per-pixel alignment residuals in the sequence to quantify the accuracy of an algorithm or set of parameters. Accuracy using several different descriptors is shown in Table 1, where single-frame Census and BRIEF descriptors are stacked over four frames for fairer comparison and 'Random' is a descriptor composed of unconstrained comparisons between pixels in the $3 \times 3 \times 4$ spacetime volume.

This evaluation method was also used to generate Figure 4, which illustrates the impact of slanted planes on reconstruction error, along with the effects of subpixel sampling in our precomputed descriptor lookup table. With slanted planes, reconstruction error is already quite low at only one sample per pixel, and there is no improvement beyond two samples per pixel. Errors are higher without slanted planes, and the improvements from denser subpixel descriptor sampling take longer to saturate.

To illustrate the advantages of using multiple input frames per output depth frame, we compared ESPReSSo to HashMatch [10] a state-of-the art, single-pattern stereo method (see Figure 5). Depthmaps generated by ESPReSSo are less noisy and more complete. We also demonstrate that using slanted planes for cost aggregation yields better results than assuming the neighborhood around a pixel is a frontoparallel plane.

[2]Model from http://hhoppe.com/proj/thesis/. See supplementary material for .ply mesh.

Descriptor	Patterns	MTAE	Outliers
Breve	1	0.99mm	7.16%
Breve	2	0.87mm	4.73%
Breve	3	0.83mm	3.71%
Breve	4	**0.81mm**	**3.07%**
Stacked Census	4	0.87mm	3.90%
Stacked BRIEF	4	**0.81mm**	3.14%
Random	4	0.84mm	3.73%

Table 1: Comparing descriptors: Mean Truncated Absolute Errors are in mm, and outliers are pixels with error above the truncation threshold (5mm).

4.1. Timings

Image regions close to the edges lie outside the frustum of the other camera, so although the raw source images are 1280×1024, we compute stereo results corresponding to an 1120×980 crop of the rectified images. The total run time on an NVIDIA Titan Xp (Pascal) is 10.39 ms, broken down by stages in Table 2.

Reference descriptors B_{ref}	0.11
Secondary descriptors B_{sec}	0.25
Compute permeability factors μ	0.13
Hypothesis generation P	0.43
Plane tests C	9.25
Connected component filter	0.13
Other output processing	0.09
Total	10.39

Table 2: Timings in ms

To minimize both resampling error and overhead, rectification is fused into the descriptor and permeability generation kernels. Plane testing is a single kernel that resamples the descriptors, computes Hamming distances, permeability filters the resulting costs across tiles, and updates winners with lowest aggregate cost. As described above, our system performs 16 iterations of both generating hypotheses and testing them; the numbers above represent the totals for all 16 rounds. The "Other output processing" line item in the Table aggregates several kernels that extract disparities and depths from the planar representation, and invalidate depth pixels as described in Section 3.4.

5. Conclusion

We have described a system and algorithms for real-time spacetime stereo reconstruction. Our approach produces high accuracy and good coverage on highly complex real-world surfaces. Because it requires no calibrated illumination sources, multiple sensors can capture depth simultane-

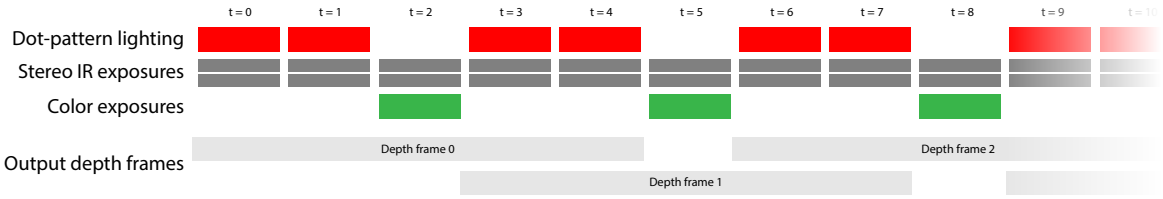

Figure 3: Frames are captured at 180Hz, and processed in overlapping groups of five frames: four with dot patterns, and one middle guide image with flood illumination. The resulting output depth frames are output at 60Hz.

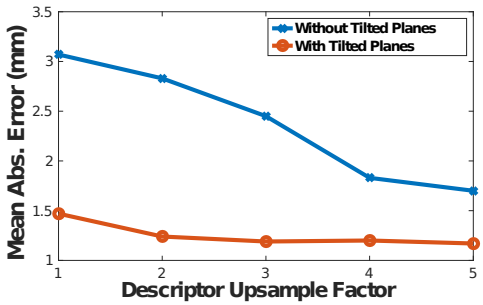

Figure 4: Slanted planes produce more accurate depth and require fewer subpixel descriptor samples.

ously without interference. (In fact, additional illumination improves accuracy by increasing dot density.) We show a spatially compact spacetime descriptor that is both discriminative and robust to motion. We also introduced a precomputation strategy for these descriptors to enable the use of slanted disparity planes, at a cost comparable to methods that are limited to aggregating costs in fronto-parallel planes, and show that this strategy gives good accuracy even for highly oblique geometry. And, unlike many fast methods that give poor answers near depth discontinuity edges, our method provides high-confidence depth very close to depth discontinuity edges.

An unusual characteristic of our method is that we employ two different types of pixel aggregation: The *breve* descriptors are fronto-parallel and spatially small, while the tile-level permeability filter aggregation has larger spatial extent, but is edge-aware. One may ask why use both types of aggregation? We find that large spatial descriptors work poorly for oblique geometry, but aggregation of single-pixel cost functions (*i.e.* pixelwise SAD and SSD) aren't discriminative enough, even when applying expensive edge-aware filters. *Breve* is both small enough to work for oblique geometry, and also big enough to be highly discriminative. Thus we find this a good tradeoff for a high accuracy real-time system.

One limitation of our present implementation is that information is not propagated between tiles as strongly as it is within a tile, so in near-planar regions, we sometimes see small depth discontinuities across tiles in the output. Such depth discontinuities can be problematic for applications that use normals or other derivative computations. But the actual depth differences are typically very small, so for example, texture reprojections from other cameras still look good at novel viewpoints (see Figure 1).

It should also be noted that our plane-picking strategy is entirely ad hoc, and we are actively investigating improvements. In particular, machine learning approaches may offer faster convergence with fewer plane tests. Likewise, we suspect learning methods could prove more effective for depth invalidation than our current heuristics.

Although our capture method is effective for moving subjects at typical human speeds, fast motion can still result in poor matches across views. Ideally, we would capture images in burst sequences of five short-exposure image pairs. We believe our algorithm will work reasonably well with shorter exposures, but is presently limited by our current camera electronics, which don't have a buffer to hold burst sequences, even when the overall bit rate remains below the USB 3 speed limit. Future work includes use of cameras with faster links to the host computer, or cameras with on-board buffering.

In spite of these limitations, we believe ESPReSSo advances the cause of high-quality real-time consumer depth acquisition. We utilize low-cost assisted infrared illumination that doesn't interfere with visible lighting and needs no calibration. Multiple such systems can be used together to capture large-scale dynamic scenes without the need for temporal fusion. Such a system can unlock many interesting interactive applications in entertainment, communications, and collaboration.

Acknowledgements: Many Googlers assisted in preparation of this work, but we especially thank Julien Valentin and Adarsh Kowdle for their HashMatch code, Brian Kuschak for IR trigger electronics, Johann Rocholl for 3D printing, Serhiy Tykhanskyy for ICP code, Chao Du for mesh rendering, T.J. Hayes for technical assistance, and Hugues Hoppe and David Gallup for proofreading.

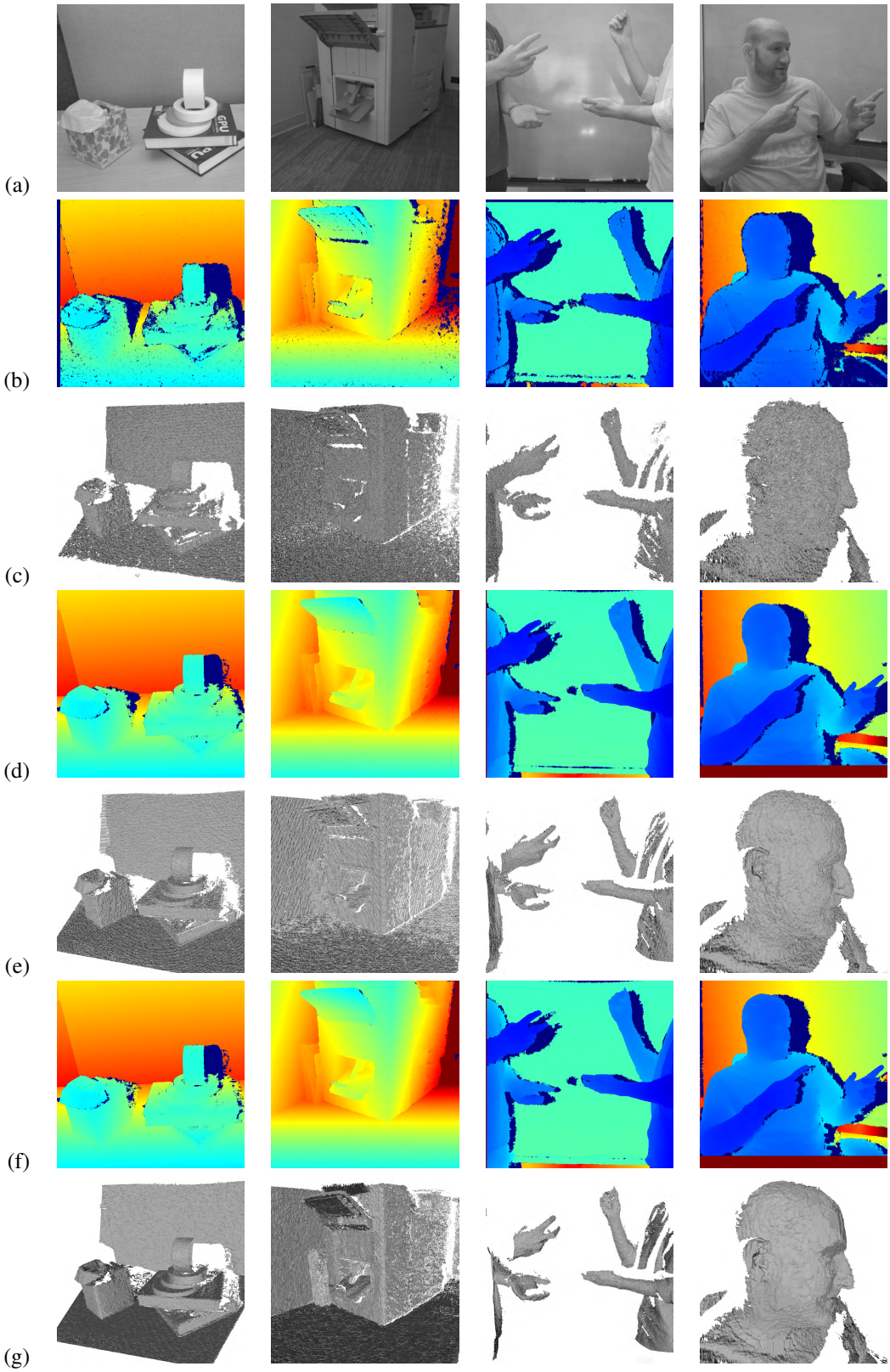

Figure 5: HashMatch [10] uses a single stereo image pair and produces reconstructions that are noisy and have poor coverage, seen in (b) depth maps and (c) meshes. Rows (d) and (e) show ESPReSSo with only frontoparallel planes and eight descriptor samples per pixel. Rows (f) and (g) show ESPReSSo using slanted planes and only two descriptor samples per pixel.

References

[1] Wikipedia: PrimeSense. https://en.wikipedia.org/wiki/PrimeSense. Accessed: 2018-05-21. 1

[2] Y. Bao, L. Tang, P. S. Schnable, and M. G. S. Fernandez. GPU-based parallelization of a sub-pixel high-resolution stereo matching algorithm for high-throughput biomass sorghum phenotyping. In *2015 ASABE Annual International Meeting*, page 1. American Society of Agricultural and Biological Engineers, 2015. 2

[3] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992. 6

[4] F. Besse, C. Rother, A. Fitzgibbon, and J. Kautz. PMBP: PatchMatch belief propagation for correspondence field estimation. *International Journal of Computer Vision*, 110(1):2–13, 2014. 2

[5] M. Bleyer, C. Rhemann, and C. Rother. PatchMatch stereo: Stereo matching with slanted support windows. In *BMVC*, January 2011. 2, 5

[6] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010. 2, 3

[7] J. Chang, J.-C. Jeong, and D.-H. Hwang. Real-time hybrid stereo vision system for HD resolution disparity map. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. 2

[8] C. Cigla and A. A. Alatan. Information permeability for stereo matching. *Image Commun.*, 28(9):1072–1088, Oct. 2013. 1, 2, 4

[9] J. Davis, D. Nehab, R. Ramamoorthi, and S. Rusinkiewicz. Spacetime stereo: a unifying framework for depth from triangulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(2):296–302, Feb 2005. 1, 2

[10] S. R. Fanello, J. Valentin, A. Kowdle, C. Rhemann, V. Tankovich, C. Ciliberto, P. Davidson, and S. Izadi. Low compute and fully parallel computer vision with HashMatch. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3894–3903. IEEE, 2017. 2, 3, 6, 8

[11] P. Heise, B. Jensen, S. Klose, and A. Knoll. Variational PatchMatch multiview reconstruction and refinement. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 882–890, Dec 2015. 2

[12] P. Heise, S. Klose, B. Jensen, and A. Knoll. PM-Huber: PatchMatch with Huber regularization for stereo matching. In *2013 IEEE International Conference on Computer Vision*, pages 2360–2367, Dec 2013. 2

[13] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(2):504–511, Feb 2013. 2

[14] Y. Li, D. Min, M. S. Brown, M. N. Do, and J. Lu. SPM-BP: Sped-up PatchMatch belief propagation for continuous MRFs. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4006–4014, Dec 2015. 2

[15] J. Lu, Y. Li, H. Yang, D. Min, W. Eng, and M. N. Do. PatchMatch filter: Edge-aware filtering meets randomized search for visual correspondence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1866–1879, Sept 2017. 1, 2, 4, 5

[16] T. Taniai, Y. Matsushita, Y. Sato, and T. Naemura. Continuous 3D label stereo matching using local expansion moves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. Early access preprint, DOI 10.1109/TPAMI.2017.2766072. 2

[17] K.-J. Yoon and I. S. Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):650–656, April 2006. 2

[18] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *Proceedings of the Third European Conference on Computer Vision (Vol. II)*, ECCV '94, pages 151–158, Berlin, Heidelberg, 1994. Springer-Verlag. 2, 3

[19] L. Zhang, B. Curless, and S. M. Seitz. Rapid shape acquisition using color structured light and multi-pass dynamic programming. In *The 1st IEEE International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 24–36, June 2002. 2

[20] L. Zhang, B. Curless, and S. M. Seitz. Spacetime stereo: Shape recovery for dynamic scenes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 367–374, June 2003. 1, 2

[21] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt, et al. Real-time non-rigid reconstruction using an RGB-D camera. *ACM Transactions on Graphics (TOG)*, 33(4):156, 2014. 2