Depth from Motion for Smartphone AR

JULIEN VALENTIN, ADARSH KOWDLE, JONATHAN T. BARRON, NEAL WADHWA, MAX DZITSIUK, MICHAEL SCHOENBERG, VIVEK VERMA, AMBRUS CSASZAR, ERIC TURNER, IVAN DRYANOVSKI, JOAO AFONSO, JOSE PASCOAL, KONSTANTINE TSOTSOS, MIRA LEUNG, MIRKO SCHMIDT, ONUR GULERYUZ, SAMEH KHAMIS, VLADIMIR TANKOVITCH, SEAN FANELLO, SHAHRAM IZADI, and CHRISTOPH RHEMANN, Google Inc.

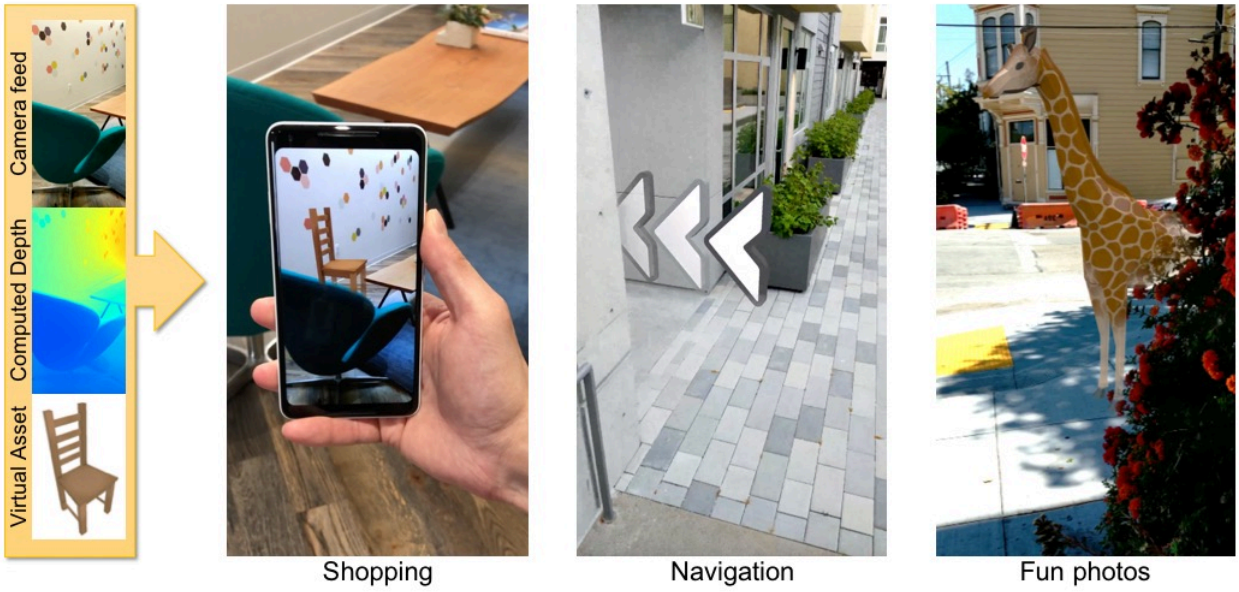


Fig. 1. AR occlusions. Estimating the depth of the scene is crucial to render virtual objects such that they realistically blend into the real context. We provide the first system capable of providing dense, low latency depth maps at 30Hz on a single mobile CPU core, using only the standard color camera found on most smartphones. Applications include AR shopping, navigation and creative photo apps.

Augmented reality (AR) for smartphones has matured from a technology for earlier adopters, available only on select high-end phones, to one that is truly available to the general public. One of the key breakthroughs has been in *low-compute* methods for six degree of freedom (6DoF) tracking on phones using only the *existing hardware* (camera and inertial sensors). 6DoF tracking is the cornerstone of smartphone AR allowing virtual content to be precisely locked on top of the real world. However, to really give users the impression of believable AR, one requires *mobile depth*. Without depth, even simple effects such as a virtual object being correctly occluded by the real-world is impossible. However, requiring a mobile depth sensor would severely restrict the access to such features. In this article, we provide a novel pipeline for mobile depth that supports a wide array of mobile phones, and uses only the existing monocular color sensor. Through several technical contributions, we provide the ability to compute low latency dense depth maps using only a single CPU core of a wide range of (medium-high) mobile phones. We demonstrate the capabilities of our approach on high-level AR applications including real-time navigation and shopping.

CCS Concepts: • **Computing methodologies** → **Computer vision**; **Epipolar geometry**; **3D imaging**; *Mixed / augmented reality*;

Additional Key Words and Phrases: depth from motion, structure from motion, motion stereo.

ACM Reference Format:
Julien Valentin, Adarsh Kowdle, Jonathan T. Barron, Neal Wadhwa, Max Dzitsiuk, Michael Schoenberg, Vivek Verma, Ambrus Csaszar, Eric Turner, Ivan Dryanovski, Joao Afonso, Jose Pascoal, Konstantine Tsotsos, Mira Leung, Mirko Schmidt, Onur Guleryuz, Sameh Khamis, Vladimir Tankovitch, Sean Fanello, Shahram Izadi, and Christoph Rhemann. 2018. Depth from Motion for Smartphone AR. *ACM Trans. Graph.* 37, 6, Article 193 (November 2018), 19 pages. https://doi.org/10.1145/3272127.3275041

Authors' address: Julien Valentin; Adarsh Kowdle; Jonathan T. Barron; Neal Wadhwa; Max Dzitsiuk; Michael Schoenberg; Vivek Verma; Ambrus Csaszar; Eric Turner; Ivan Dryanovski; Joao Afonso; Jose Pascoal; Konstantine Tsotsos; Mira Leung; Mirko Schmidt; Onur Guleryuz; Sameh Khamis; Vladimir Tankovitch; Sean Fanello; Shahram Izadi; Christoph Rhemann, Google Inc.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2018 Copyright held by the owner/author(s).
0730-0301/2018/11-ART193
https://doi.org/10.1145/3272127.3275041

1 INTRODUCTION

In recent years, the mobile industry has significantly invested in augmented reality (AR) for smartphones. Your mobile phone can now be a viewfinder on an augmented world, where virtual content is rendered on top of the color camera feed in real-time. Almost all emerging high-end or mid-range phones now have some form of six degree of freedom (6DoF) tracking capability using just the typical sensors found inside these devices – the color camera and inertial measurement unit (IMU). This shift has required many breakthroughs in visual inertial odometry (VIO) and simultaneous localization and mapping (SLAM).

These advancements have led to the release of sparse 6DoF tracking platforms such as ARKit [Apple 2018] and ARCore [Google 2018] with AR applications that create the illusion that users can place and lock virtual objects in their environments. However, the illusion breaks as soon as visual inconsistencies appear, such as wrong occlusions between real and virtual objects.

To achieve another level of immersion, one needs dense depth maps on device, and in real-time. Depth is a prerequisite for more realistic AR, including correct handling of occlusions of virtual content by real objects, better placement of virtual content, and enabling interactions (such as physics collisions) between real and virtual content. However, despite many breakthroughs on smartphone AR, none of these devices can currently provide real-time dense depth maps without adding a dedicated new sensor. While we are clearly heading towards a future where dedicated depth sensors will become more ubiquitous, for now, adding such sensors means that a lot of the ubiquity and appeal of smartphone AR is lost, with additional negative impact on cost, power, and industrial design. Therefore, leveraging monocular color cameras is the best path to scale dense depth estimation to millions of existing devices. Although the literature on monocular depth estimation is extensive (see related work), no method exists that is capable of providing dense and edge-preserving depth maps at low computation on mobile phones today.

This paper introduces a novel pipeline capable of supplying dense and low latency QVGA depth maps at 30Hz, leveraging only a single RGB camera and a single CPU core of a smartphone. Figure 1 illustrates how this depth map can be used to realistically render virtual objects in real-time, enabling new effects that can enhance a wide array of mobile applications including shopping, street navigation and self expression.

The task at hand has several challenges. First, our approach needs to work across a variety of different phones with different camera sensors for which we do not have control over parameters like exposure or focus. Second, we must deliver dense depth at low latency and low computation to the user, even under conditions of poor tracking or untextured environments. Third, we want to support mid-tier mobile CPUs—e.g. single 1.9GHz Qualcomm A53 CPU core.

We solve this task through a novel depth from motion pipeline, comprising of the following technical contributions:

- A procedure allowing the use of polar rectified images for efficient stereo matching.
- A new keyframe selection strategy.
- A highly optimized stereo matching algorithm leveraging the best aspects of PatchMatch Stereo [Bleyer et al. 2011] and HashMatch [Fanello et al. 2017a].
- New extensions of the bilateral solver [Barron and Poole 2016] for depth post-processing that: (a) lead to higher quality point clouds by encouraging planar solutions, (b) a new initialization scheme leading to faster convergence and lower computation requirements, and (c) a novel formulation for producing temporally stable results.
- Finally, a new late stage rendering step that provides a fluid low-latency experience to users.

2 RELATED WORK

Mobile Depth. We focus on related work for estimating depth on mobile phones using existing monocular sensors, as opposed to dedicated hardware. For a more general discussion about the state of the art on passive depth estimation, we refer the interested reader to [Fanello et al. 2017a; Hamzah and Ibrahim 2016; Kendall et al. 2017; Scharstein and Szeliski 2002].

In the literature, passive depth estimation on mobile platforms has almost exclusively been studied in the context of 3D reconstruction. These types of algorithms fuse *sparse* depth maps over time into an underlying volumetric representation. Notable methods that use Truncated Signed Distance Functions (TSDF) as a representation include [Kähler et al. 2015; Ondrúška et al. 2015; Schöps et al. 2017b].

The approach described in [Ondrúška et al. 2015] generates sparse depth maps of size 320×240 at 50Hz on an iPhone GPU using an approximation of PatchMatch Stereo [Bleyer et al. 2011] and a dense tracking pipeline akin to DTAM [Newcombe et al. 2011] but with IMU initialization. Their overall pipeline is distributed across the CPU and the GPU and runs at 25Hz, but is unfortunately unable to handle general camera motion. The pipeline of [Schöps et al. 2017b] computes sparse 320×240 depth maps at 12Hz on the GPU of Tango Tablets. These depth maps are then integrated in a TSDF at 8Hz. [Kähler et al. 2015] describe a pipeline that fuses depth maps at 24Hz on an iPad Air 2, but assumes that the depth is provided by a third party.

Unfortunately, the use of mobile GPUs for continuous mobile usage is prohibitive, due to power consumption, thermal considerations (GPUs cause thermal issues if continually maxed), and the need to have such GPU resources available for rendering. TSDF representations also generally require significant amounts of memory and also need large camera motions to carve away edge-fattening caused by stereo-matching and to fill holes. Surfels are less memory demanding than a TSDF and are used by [Kolev et al. 2014] to fuse depth maps generated by the technique described in [Tanskanen et al. 2013], which takes seconds per frame.

Another approach is described in [Schöps et al. 2014] where 320×240 semi-dense depth maps are computed at 15Hz on a mobile CPU, from which a collision mesh is extracted. The mesh does not cover the whole image however, and object boundaries are not explicitly handled. Therefore, triangles can span objects belonging to the foreground and background of the scene.

Fusion-based methods aside, [Suwajanakorn et al. 2015] estimate depth from defocus. Given a focal sweep captured by a moving

camera, this technique compensates for small viewpoint changes that occur during the acquisition of the focal stack and performs auto-calibration and depth estimation. Although a mobile phone is used for the data capture, the inference process takes 20 minutes per depth map.

Various deep learning-based formulations to depth estimation have been recently investigated. [Liu et al. 2016] trained unary and pairwise MRF potentials using CNNs to predict depth from a single image in 120*ms*. A two-scale deep network was proposed in [Eigen et al. 2014] and trained on images and their corresponding depth maps. [Garg et al. 2016] minimize a reconstruction loss using a warp image for single view depth estimation, but needs to linearize their objective using Taylor expansion to backpropagate through it. [Godard et al. 2017] take a similar approach, but instead use bilinear sampling for a fully differentiable objective function. Although [Eigen et al. 2014] and [Godard et al. 2017] report inference at $< 35ms$ per frame, they both still need a high-end desktop GPU to achieve this real-time performance.

Late stage rendering. To reduce perceived latency, late stage rendering or time-warping [Evangelakos and Mara 2016; Van Waveren 2016] can be used. It is mostly studied in the context of AR/VR rendering where head-motion occurs after rendering and needs to be compensated for prior to display to reduce perceived latency. For best user experiences, this end-to-end latency [Mine and Bishop 1993] should not exceed 20ms [Zhang and Luo 2012].

Common approaches apply a rotational-only correction for asynchronous reprojection by applying a homography which transforms the rendered viewpoint to the final display viewpoint [Hartley and Zisserman 2003]. Fully positional-aware warping solutions require access to perfect depth of the scene, which is feasible for rendered content, but more difficult in the general case. In this paper, we present a novel screen-space technique to reduce latency that does not rely on accurate depth estimation or simplified scene assumptions.

Depth map densification. Most stereo or depth-recovery algorithms contain a *densification* step in which noisy, sparse, or otherwise incomplete depth observations are turned into a dense and smooth depth map, often using a "reference" RGB image to encourage depth edges to co-occur with color edges. For an extensive overview of the literature, we refer the interested reader to [Pan et al. 2018; Park et al. 2014; Weerasekera et al. 2018].

Variational inpainting [Oliveira et al. 2001] approaches based on Total Variation (TV) [Shen and Chan 2002] have been shown to run in real-time on the GPU of a tablet [Schöps et al. 2017a]. However, these systems have significantly more computational performance than a CPU core of a phone, while carrying the challenges associated with mobile GPUs outlined earlier.

The fast bilateral solver [Barron and Poole 2016] is another approach to denoise and complete depth maps. It produces high-quality results by solving a global optimization problem in "bilateral space" as opposed to over all pixels in an image, resulting in runtimes that are largely independent of image resolution. The bilateral solver has been used successfully on mobile devices as shown in [Wadhwa et al. 2018] and recently received a "hardware-friendly" extension described in [Mazumdar et al. 2017] that allows it to more efficiently leverage parallelization and vectorization on mobile hardware. We build upon this work in this paper.

Although there is a significant body of work on using deep learning to inpaint color images (e.g, [Oord et al. 2016; Pathak et al. 2016], the literature on deep depth inpainting is more recent. Given aligned color and depth input, [Zhang and Funkhouser 2018] predict surface normals and occlusion boundaries from the color and then solve a linear system to inpaint the depth. Their approach runs at 1.8 seconds per frame on a Titan X GPU, making it prohibitive for mobile use cases.

Depth map temporal filtering. To achieve a temporally coherent depth map many works incorporate temporal consistency directly into the stereo matching process. The works most closely related to our approach are [Hosni et al. 2011; Richardt et al. 2010] where a spatio-temporal filter based on the guided image filter and bilateral grid are used to smooth a cost volume. Other approaches use spatio-temporal filters to post-process the depth map to provide a more coherent solution [Richardt et al. 2012]. However, these approaches perform the filtering on a GPU. We opt for incorporating temporal consistency into the bilateral solver which allows for a computationally efficient and effective approach.

3 SYSTEM OVERVIEW

We now outline all the components of the proposed depth from motion pipeline.

As the user navigates through their environment with a smartphone in hand, our pipeline starts by tracking 6DoF poses using the off-the-shelf VIO platform of ARCore [Google 2018]. Note, that our system could use any other VIO or SLAM platform at this stage.

Once the the 6DoF tracking pipeline is initialized and given the latest available camera image (we use grayscale images for computational reasons), the first step towards computing a depth map consists of identifying a keyframe from the past image frames that is suitable to perform stereo matching.

Next, the relative 6DoF pose between the keyframe and the current frame is used to perform polar rectification. Stereo-rectification is not a mandatory step, but it significantly speeds up stereo matching by reducing the correspondence search to the same horizontal lines in both images.

A very fast CRF solver is used to compute correspondences; wrong estimates are discarded via an efficient machine learning based solution, leading to disparity maps that are almost free of outliers. From disparities, one can estimate a sparse depth map through triangulation.

The sparse depth map is then fed into a novel variant of the fast-bilateral solver [Barron and Poole 2016] that generates a bilateral grid of depth (as opposed to a depth map). The bilateral grid can be converted on-demand into a dense, spatio-temporal smooth depth map. We do this by slicing the grid with the *most recent available image* (as opposed to the frame used to populate the bilateral grid) which ensures that the edges of the produced depth map are aligned with the RGB image currently displayed on the smartphone.

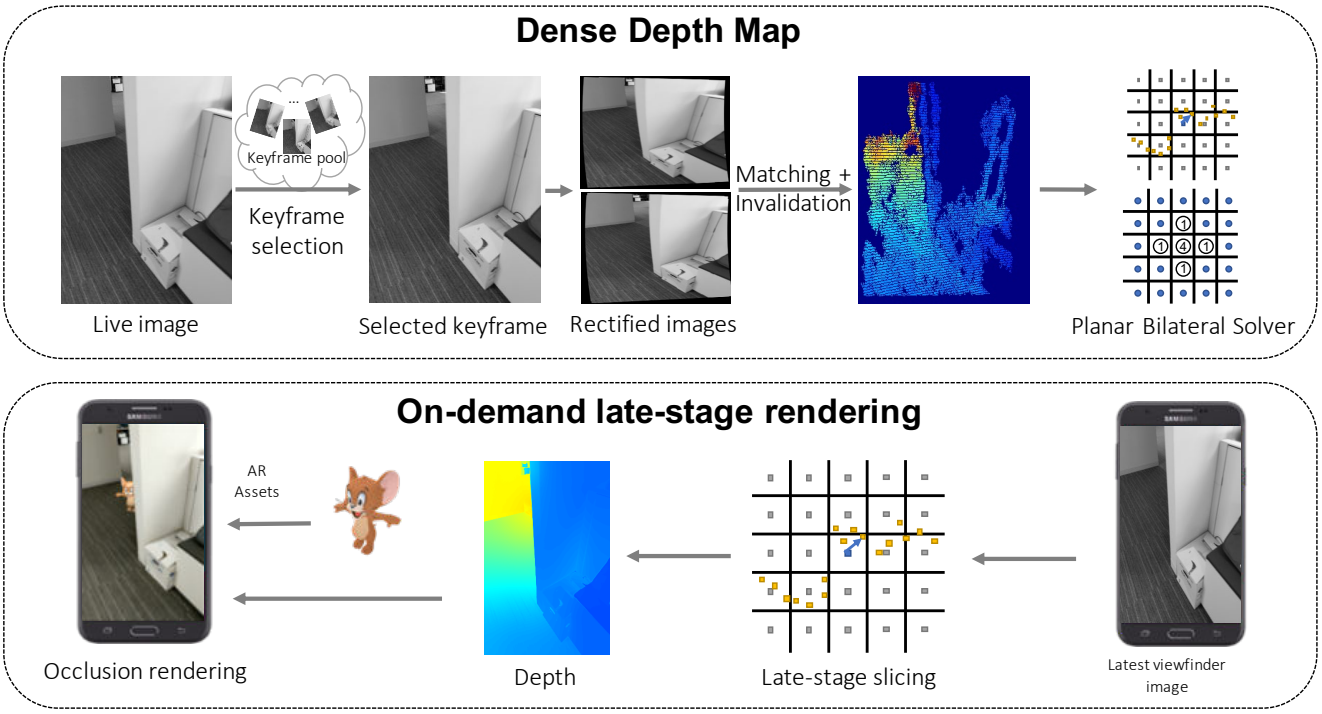


Fig. 2. Depth from Motion pipeline. Dense depth map estimation (generation of the bilateral grid) and late stage rendering using the latest viewfinder image (conversion of the bilateral grid into a dense, spatio-temporal smooth depth map) are running on different threads, allowing to provide depth with very low latency.

The generation of the bilateral grid and the slicing are decoupled and run on separate threads. Depth maps can therefore be generated at high frame-rate with very low latency, efficiently making it *independent* of the run-time of the CRF inference, allowing deployment of our system on mid and high-end devices without sacrificing quality or effective framerate.

Finally, high level applications can leverage this real-time depth-estimation to enable effects such as AR occlusions, as shown in Figure 1. Figure 2 illustrates the main steps of the above-described pipeline, which we detail in the following sections.

4 KEYFRAME SELECTION

Our approach for depth estimation is based on stereo matching between the most recent image and a past keyframe. The choice of the keyframe is dependent on several disparate factors, each of which contribute to the potential matching quality of a candidate keyframe. For instance, greater depth accuracy is gained by increasing the stereo baseline between the chosen keyframe and the present position, but such frames are also further back in time, which can introduce temporal inconsistencies.

Previous approaches for motion stereo often rely on keyframes from a fixed time delay [Kim et al. 2016; Zhou et al. 2017]. This method is robust when under constant movement, such as in a vehicle, but usage for handheld devices results in sporadic uneven motion. Other methods rely on feature or geometry tracking, rather than matching correspondences to a specific frame [Karsch et al. 2016; Li et al. 2006], but these algorithms often result in sparse depth or are unable to run at adequate framerates. [Pradeep et al. 2013] select the optimal keyframe for stereo matching in terms of baseline and image overlap. Finally, [Schönberger et al. 2016; Zheng et al. 2014] introduce notable techniques for pixel-wise view selection, but these approaches are unfortunately computationally too demanding for mobile scenarios.

In this paper, we define a soft-cost function to select the optimal keyframe for the latest target frame using several metrics. We keep a fixed-capacity pool of potential keyframes, with each newly captured frame being added to this pool if the 6-DoF visual-inertial tracking is successful, replacing outdated frames. The cost metrics we use to pick the best keyframe from the pool are:

- $b_{i,j}$: The baseline distance in 3D between two frames i and j. We want this value to be large.
- $a_{i,j}$: The fractional overlap, in the range $[0, 1]$, of the image areas for the frames i and j, computed based on their viewing frustums. We want to maximize this value.
- $e_{i,j}$: The measured error of pose-tracking statistics for the two frames. We want to keep this value small.

We use a multidimensional cost function to select a keyframe k to pair with the latest reference image r, producing the minimum total cost from the keyframe pool K:

$$\underset{k \in K}{\operatorname{argmin}} \quad \frac{\omega_b}{b_{r,k}} + \omega_a(1 - a_{r,k}) + \omega_e e_{r,k} \tag{1}$$

The choice of the weighting ω_b for the baseline is relative to the nominal desired baseline for the target scene depth. Choosing candidate frames based on a known baseline is a classic technique for motion stereo [Jain et al. 1987; Nevatia 1976], but for modern mobile systems this metric must also be weighted against other considerations. In practice we set $\omega_b = 0.4$, with a strict limit of minimum allowed baseline of 4 cm.

The cost term for area overlap $a_{r,k}$ is weighted heavily with ω_a, since although successful matching can occur between frames with only a partial overlap, greater overlap reduces the need of depth values to be extrapolated to other parts of the target image. We set $\omega_a = 0.8$, with a strict threshold of minimum allowed overlap at 40%.

If 6-DoF motion tracking–e.g. ARKit or ARCore–produces a frame that is measured to have poor confidence, then the frame is never added to the keyframe pool at all. However, even candidate keyframes with high confidence may have some relative error to the latest reference frame. This relative error cost is weighted against the other costs with ω_e. This weight is set at $\omega_e = 0.5$, with a strict thresholds ensuring the measured velocity variance between the chosen keyframe and the latest reference frame does not exceed $5{\times}10^{-4}$ m/s, as well as ensuring the measured acceleration bias does not exceed $0.2\,m/s^2$.

The above parameters were chosen by maximizing pixel-wise depth error and minimizing the number of reported invalid pixels across a set of test datasets. The choice of these weights improved depth accuracy by about 10%. Now that a keyframe has been identified, we next perform stereo rectification.

5 STEREO RECTIFICATION

Given two cameras and their respective poses, one can estimate the fundamental matrix [Hartley and Zisserman 2003] that governs how pixels corresponding to the same 3D point are related. For two corresponding pixels x and x' that come from the projection of a 3D point X, and the fundamental matrix F, one can observe that x' lies on the line $l' = Fx$. The correspondence search is then constrained to a one-dimensional problem. Once correspondences are estimated, for instance using the matching algorithm described in Section 6, the depth of any given pixel can be estimated through triangulation.

Due to the linear cache pre-fetch behavior of modern CPUs, it is significantly more efficient to perform this 1-d search along horizontal lines in images that are stereo-rectified. Such images have the property that all epipolar lines are parallel to the horizontal axis of the image. Standard dual-camera setups with a fixed baseline have the advantage that at each frame, the captured images are already close to be stereo-rectified. Mostly due to mechanical imprecision, these images still need to be stereo-rectified in software. For these setups, practitioners usually resort to using planar rectification [Faugeras et al. 2001; Loop and Zhang 1999] due to the availability of production-level public implementations [Bradski and Kaehler 2000].

When dealing with a single camera that is freely moving, epipoles can be anywhere on the image plane. In particular, when the user is moving forward with their camera, the epipole is inside the image, which causes traditional techniques such as planar rectification to fail. In this paper, we aim at allowing users to freely move with their mobile phones and opt for using the polar rectification technique described in [Pollefeys et al. 1999].

5.1 Using polar rectified images for stereo matching.

In [Pollefeys et al. 1999], the authors describe an algorithm that, given a pair of images and their relative pose, transforms these images such that their epipolar lines are parallel and corresponding epipolar lines have the same vertical coordinate. Also, for computational reasons described earlier, it is generally desired that for a point (x, y) in the left-rectified image, the correspondence lies at (x', y) in the right-rectified image with $x' < x$ and $\tau_{min} < x - x' < \tau_{max}$, with τ_{min} a small constant, usually 0 or 1, and τ_{max} the maximum disparity, 40 in our case. [Pollefeys et al. 1999] does not describe how to obtain rectified images for which corresponding pixels lie in a *fixed* disparity range. Conversely, we propose a solution that constrains solutions to a known disparity interval which can be efficiently exploited while estimating correspondences.

Estimating image-flip. As illustrated in Figure 3, some camera configurations can lead to rectified images that, modulo disparity, are flipped version of each other. These configurations break the requirement of $x' < x$. These cases are detected when the dot products between the ray through the optical centers and their respective image center in world coordinates, and the vector that links both optical centers, have opposite signs. In that case, one of the images needs to be flipped.

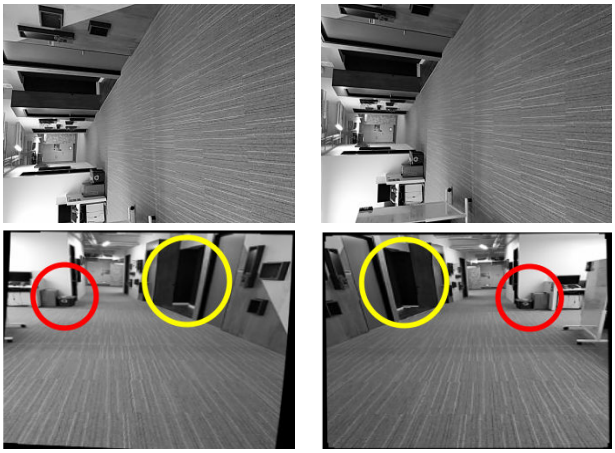


Fig. 3. **Image flip.** Notice how the configuration of the input images (top) lead to rectified images that are flipped versions of each other. Indeed, in the bottom left image, the red circle is on the left side of the yellow circle, where the situation is flipped in the bottom right image.

Estimating image-swap and x-shift. As mentioned earlier, efficient stereo matching assumes that all correspondences reside in a predefined range of disparity values $[\tau_{min}, \tau_{max}]$, with $\tau_{min} > 0$. This assumption can be violated with negative disparities, illustrated in Figure 4, or disparities that exceed the maximum disparity assumed by the stereo-matcher. We therefore apply a shift to bring disparities into a valid range.

First, given the relative transform between the two frames and a pre-defined range of depth values $[D_{min}, D_{max}]$ the system is expected to handle, we estimate $[\tau^L_{min}; \tau^L_{max}]$ and $[\tau^R_{min}; \tau^R_{max}]$. These intervals correspond to the disparity ranges required to make predictions if the left or right image is used as reference, respectively. We then select the reference image and the amount of horizontal shift required to fit in the expected disparity range $[\tau_{min}, \tau_{max}]$ based on which configuration requires the smallest shift and on the sign of the estimated disparities. In the event that the expected range of disparities is too small to accommodate the current pair, one can re-size the rectified images accordingly.

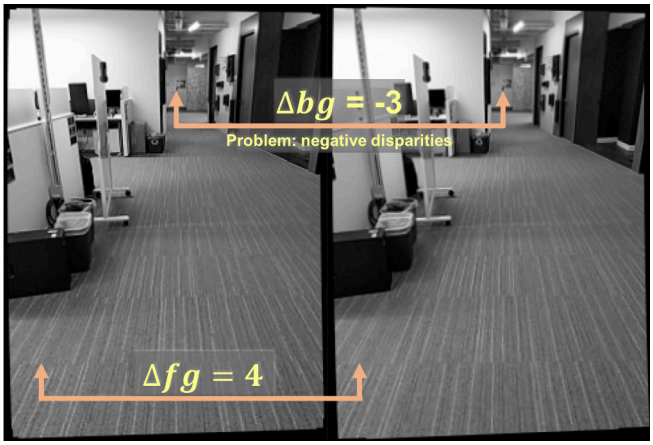


Fig. 4. **Horizontal shift.** In this configuration, we have Δfg = 4, Δbg = -3. Efficient stereo-matchers assume that candidate disparities range between 0 and a pre-defined maximum disparity. By default, this assumption can be violated when using polar-rectified images.

Improving horizontal resolution for higher quality stereo-matching results. When operating on textured scenes, standard techniques like PatchMatch Stereo [Bleyer et al. 2011] or HashMatch [Fanello et al. 2017a] have a very similar sub-pixel accuracy around 0.2 pixels. By default, the rectified images can be significantly bigger that the original image, which is not a desirable property for computational reasons. A naive strategy would be to re-size the rectified images to the original resolution, which can lead to rectified images with a width smaller than it could be, directly impacting depth-accuracy due to the fixed sub-pixel accuracy described above. Hence, to increase depth-accuracy, we propose to re-size the rectified images such that the size of $[\tau_{min}; \tau_{max}]$ matches the maximum expected number of disparities (40 in our case), while keeping the total number of pixels constant.

Accounting for some pose uncertainty. Most imprecision in the relative transform between the two images lead to gross rectification problems. In practice, systems like ARCore only suffer from minor pose imprecision. When the epipoles are 'far' from the images, some small pose imprecision lead to imprecision in the position of the epipole, which in turn can lead in solving for both negative and positive disparities. We found that setting τ_{min} between $[5, 10]$ as opposed to 0 or 1 effectively combats small pose inaccuracies. When the epipole(s) are located within the image, we preemptively invalidate a disk of pixels located around the epipole. In our system the radius of this disk is 20 pixels.

At this stage, regardless of the trajectory that the smartphone has followed, we have a rectified pair of images. The next section addresses performing stereo-matching on these rectified images.

6 STEREO-MATCHING

Recent works [Fanello et al. 2017b] and [Fanello et al. 2017a] have demonstrated high-resolution stereo-depth estimation respectively at 500Hz and 1000Hz on a high end GPU. In the following, we briefly describe the paradigm on which these techniques are based on.

Assuming the likelihood of solutions are well captured by a conditional probability from the exponential family

$$P(Y|D) = \frac{1}{Z(D)} e^{-E(Y|D)}, \tag{2}$$

practitioners often favor a factorization of the form

$$E(Y|D) = \sum_i \psi_u(y_i = l_i) + \sum_i \sum_{j \in N_i} \psi_p(y_i = l_i, y_j = l_j) \tag{3}$$

which is usually referred to as a pairwise-CRF. Here $Y := \{y_1 \dots y_n\}$ is the set of latent variables associated to pixels $\{x_1 \dots x_n\}$. Each $y_i \in Y$ can take values in $\mathcal{L}$, which is a subset of $\mathbb{R}$ and corresponding to disparities. Finally, N_i is the set of pixels adjacent to pixel i. Although NP-hard to solve in general [Boykov et al. 2001], this decomposition has been widely used for numerous computer vision tasks. The term ψ_u is usually referred to as the unary potential, and in the context of depth estimation via stereo matching, measures the likelihood that two pixels are in correspondence. The function ψ_p is commonly referred to as the pairwise potential and acts as a regularizer that encourages piece-wise smooth solutions. The components that make the minimization costly are evaluating ψ_u and the number of steps that the chosen optimizer takes to converge to good solutions. In this work, we use the unary potential described in [Fanello et al. 2017a] for its low-computational requirements, a truncated linear pairwise potential as pairwise potential. We optimize this cost function using a hybrid of PatchMatch [Barnes et al. 2009] and HashMatch [Fanello et al. 2017a] that is particularly efficient on CPU architectures, which we introduce below.

6.1 Vectorized Inference

The basic idea of CRF inference using PatchMatch [Barnes et al. 2009] is to propagate good labels to neighboring pixels, exploiting local smoothness of solutions. Different propagation/update strategies have been explored in the literature, some of them designed for GPU architectures [Bailer et al. 2012; Fanello et al. 2017a; Pradeep et al. 2013].

In this section, we introduce a propagation strategy that is tailored for modern CPU instruction sets that rely on Single Instruction Multiple Data (SIMD) instructions to achieve intra-register parallelism and improved throughput. Achieving high performance on such architectures, mainly ARM NEON for mobile devices, requires efficient utilization of these vector registers. In particular, loading data from memory benefits from use of *coherent loads* - that is, for a vector register with n lanes of width b bits, SIMD instruction set

architectures (ISAs) typically offer an instruction that loads $n * b$ sequential bits, filling all lanes with one instruction. The counterpart is a *diverged load*, where each lane of the vector register is inserted into the vector register one at a time. Such *diverged loads* are required when the memory to be loaded is not sequential. This behavior poses a challenge to stereo algorithms, which typically samples diverged offsets per pixel when exploring the solution space. To make matters worse, this data parallelism is inherently directional. For typical image layouts in memory, a vector register of pixels represents a subset of a particular image row, which prevents typical inference schemes (e.g. PatchMatch) from mapping well to SIMD architectures. The propagation of information horizontally in the image prevents efficient utilization of vector registers.

Recently, [Fanello et al. 2017a] has demonstrated an inference technique that allows updates to each pixel in parallel, allowing depth map prediction at 1000Hz on GPU. Compared to PatchMatch, HashMatch requires more iterations for information to propagate further in the image, but each iteration is substantially cheaper and can be performed independently. PatchMatch, by contrast, is completely sequential in nature: it iteratively goes from one pixel to the next, evaluates some particle/solutions for that pixel, and continues until reaching the end of the image. Motivated by the strengths of each approach, we propose a hybrid variant that is well-suited for SIMD architectures. Instead of performing multiple independent propagation passes for each of the eight directions, we perform k passes in sequence, each designed to utilize the data-parallelism of the underlying vector architecture. For typical scenes, k ranging from 2 to 4 is sufficient. During even-numbered passes, each pixel considers hypotheses from the three neighbors above it (that is, the pixel at (x, y) considers hypotheses from $(x-1, y-1), (x, y-1), (x+1, y-1)$) in addition to the currently stored hypothesis. During odd-numbered passes, each pixel considers hypotheses from the three neighbors below it in addition to the currently stored hypothesis. Rows are processed sequentially, starting at the top of the image in even-numbered passes and the bottom of the image in odd-numbered passes. Consequently, all pixels for a given row are independent of all other pixels in the same row, allowing parallel processing. Is is important to note that NEON acceleration in the inference step is possible due to breaking of dependency chains in the X direction (rows). The reason for this is that vector units of SIMD processors load data in chunks along the X direction, and hence they are more efficient at vectorizing operations on pixels at discrete Y values. As is standard for such inference strategies, we evaluate each hypothesis by summing the stereo matching unary cost and a weighted smoothness term, the aforementioned truncated linear pairwise potential. The unary cost evaluation cannot be fully parallelized due to the distinct disparity value in each lane of the vector register. However, we can fully parallelize the rest of the data movement: the load of initial disparity values, the load of neighboring disparity values, and the smoothness cost computation. Overall, the proposed approach has been consistently measured as 4x faster than HashMatch and 10x faster than PatchMatch.

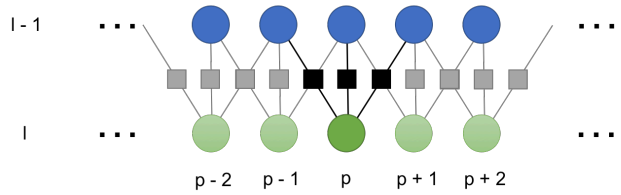


Fig. 5. The proposed factorization makes any pixel belonging to a given horizontal line (e.g. green pixels) to be independent to each other. If we now focus on pixel p, one can update it's prediction by evaluating the particle currently at p, and the particles in the neighbors of p (black link); the updated solution is the particle that minimizes this local CRF. Best viewed in color.

6.2 Invalidation

The approximate MAP inference performed over the pairwise conditional random field yields one disparity value estimated for each pixel in the image. Unfortunately, when the scene lacks texture (e.g. white wall in Figure 7, second line) or contains repetitive patterns, the MAP solution of the corresponding pixels can be wrong. It is usually preferred to invalidate such pixels since the distribution of their errors depends on image content, and hence estimating the MLE of these distributions is non-trivial (e.g. to be used in KinectFusion-like filtering). Invalidation is usually performed using thresholding on the unary cost of the solution [Fanello et al. 2017b; Mühlmann et al. 2002], using left-right consistency check [Mühlmann et al. 2002; Scharstein and Szeliski 2002], using connected component analysis [Fanello et al. 2017b], or a combination of the above. Performing a left-right consistency check leads to good invalidation results, but involves computing a disparity map for each image, which adds a significant computational cost to the pipeline.

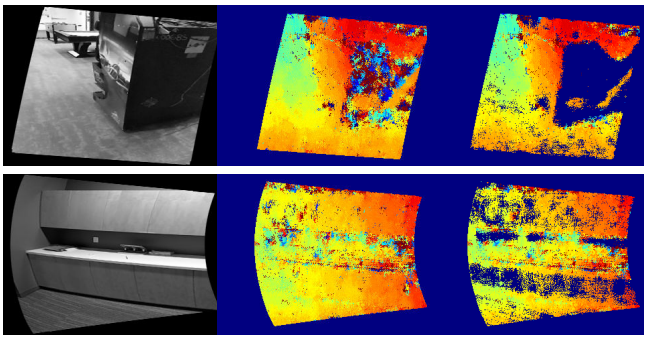


Fig. 6. CRF-cost invalidation in disparity space. (a): Rectified image. (b): Raw output from the matcher. (c): CRF-cost invalidation.

By breaking the whole CRF in cliques containing each pixel and their immediate neighbours, one can compute the negative log-likelihood $\mathcal{L}_i$ using:

$$\mathcal{L}_i = \psi_u(l_i) + \sum_{j \in \mathcal{N}_i} \psi_p(l_i, l_j) \tag{4}$$

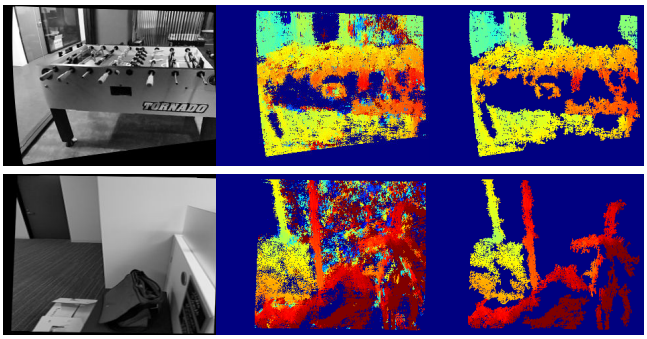


Fig. 7. Connected component invalidation in disparity space. (a): Rectified image. (b): CRF-cost invalidation. (c): Connected component invalidation.

This formulation leads to slightly better invalidation results than only considering the unary potential. As can be observed in Figure 6, pruning unlikely solutions removes a large portion of undesirable pixels. Unfortunately, in the case of untextured regions, the likelihood of solutions can be high yet incorrect. Following prior art [Fanello et al. 2017b], we invalidate small connected components in disparity space. The resulting depth map is free from the vast majority of unstable predictions, as can be observed in Figure 7. Depending on the compute architecture, running this last invalidation step can become as expensive as solving the CRF described in Equation 3. We approximate the connected component invalidation step using a single decision tree to minimize the computational resources required [Criminisi et al. 2012]. In particular, we model this invalidation step as a classification problem, where we assign to valid pixels a positive label $y = 1$ and for invalid pixel a label $y = -1$. In the context of predicting the confidence of time of flight, the authors of [Reynolds et al. 2011] also propose to use Random Forest. Unfortunately, their approach runs in 5s on 200×200 frames. For the sake of completeness, we briefly describe the training procedure for decision trees. A decision tree consists of split nodes and leaves. Each split node n stores a 'weak learner' that is parameterized by function parameters θ_n and a scalar threshold τ_n. To perform inference over the tree for pixel p, one starts at the root of the tree and evaluates:

$$s(p, n) = \mathbb{1}[f(p, \theta_n) > \tau_n], \tag{5}$$

If $s(p, n)$ evaluates to 0, the inference continues over the left children of node n, and over the right children otherwise. This process repeats until reaching a leaf, which contains a binary probability distribution over the prediction space, invalidation in this case.

As it is common practice, we chose f to be a dot product between the values of two pixel indices located around p. The values of θ_n and τ_n are greedily optimized to maximize the information gain:

$$IG(\theta_n, \tau_n) = E(\mathcal{S}) - \sum_{c \in \mathcal{L}, \mathcal{R}} \frac{|\mathcal{S}_c|}{|\mathcal{S}|} E(\mathcal{S}_d), \tag{6}$$

with E the Shannon entropy, $\mathcal{L}, \mathcal{R}$ the left and right children of node n. Finally, each leaf stores the probability for a pixel to be valid or invalid.

At test time, inference over that tree is performed for each pixel, allowing to decide which pixels should be invalidated.

6.3 Disparity to depth

Now that we have an outlier-free disparity map, we can finally infer depth. When one uses planar rectification, given a disparity d, a baseline b, and a focal length f, the depth Z can be trivially computed as $Z = \frac{bf}{d}$. However, this closed form solution cannot be used in the case of polar rectification.

A well known solution in literature to deal with these cases is the optimal triangulation method proposed in paragraph 12.2 of [Hartley and Sturm 1997]. Optimal triangulation requires minimizing a polynomial of degree 6, which could be inefficient on mobile architectures. Therefore we resort to solving the simpler linear problem described in [Hartley and Zisserman 2003], which is not optimal but fast to solve.

7 BILATERAL SOLVER EXTENSIONS

The previous sections describe how to obtain depth maps with few false positives. However, these depth maps are sparse (containing information only in textured regions), temporally inconsistent, and are not aligned with the edges in the image. In this section we use a novel extension of the bilateral solver to efficiently generate dense, temporally stable, and edge aligned depth maps with low latency.

Before modifying the bilateral solver, we first review it as it has been previously described in the literature. We build upon the "hardware friendly" variant of the solver as presented in [Mazumdar et al. 2017], which is built upon the original bilateral solver as described in [Barron and Poole 2016], which itself builds upon the optimization approach of [Barron et al. 2015].

The bilateral solver is defined as an optimization problem with respect to a "reference" image $\mathbf{r}$ (in our case, a grayscale image from the camera), a "target" image $\mathbf{t}$ of noisy observed values (in our case, a noisy depth map as computed in Section 6), and a "confidence" image $\mathbf{c}$ (in our case, the inverse of the invalidation mask as defined in Section 6.2). The solver recovers an "output" image $\mathbf{x}$ that is close to the target where the confidence is large while being maximally smooth with respect to the edges in the reference image, by solving the following optimization problem.

$$\underset{\mathbf{x}}{\text{minimize}} \ \frac{\lambda}{2} \sum_{i,j} \hat{W}_{i,j} \left(x_i - x_j\right)^2 + \sum_i c_i (x_i - t_i)^2 \tag{7}$$

The first term encourages that all pairs of pixels (i, j) to be smooth according to the bilateral affinity matrix $\hat{W}_{i,j}$ and the smoothness parameter λ (set to 4 in our experiments), while the second term encourages each x_i to be close to each t_i when c_i is large. The $\hat{\mathbf{W}}$ matrix is a bistochastic version of a bilateral affinity matrix $\mathbf{W}$, where each $W_{i,j}$ is defined as:

$$W_{i,j} = \exp\left(- \frac{\left(p_i^x - p_j^x\right)^2 + \left(p_i^y - p_j^y\right)^2}{2\sigma_{xy}^2} - \frac{\left(r_i - r_j\right)^2}{2\sigma_r^2} \right) \tag{8}$$

where for each pixel i, (p_i^x, p_i^y) is its (x, y) location and r_i is its grayscale intensity in our reference image $\mathbf{r}$. These spatial and intensity dimensions are modulated by bandwidth parameters σ_{xy} and σ_r, which we both set to 16 in our experiments.

As shown in [Barron and Poole 2016], the large and dense bistochastic bilateral affinity matrix can be represented with a compact factorization using a bilateral grid:

$$\hat{\mathbf{W}} = \mathbf{S}^{\mathrm{T}} \operatorname{diag}\left(\frac{\mathbf{n}}{\mathbf{m}}\right) \mathbf{B} \operatorname{diag}\left(\frac{\mathbf{n}}{\mathbf{m}}\right) \mathbf{S} \tag{9}$$

Where the $\mathbf{S}$ and $\mathbf{S}^{\mathrm{T}}$ matrices splat and slice into a bilateral grid respectively, $\mathbf{B}$ is a $[1, 2, 1]$ blur along the three dimensions of a bilateral grid, and vectors $\mathbf{m}$ and $\mathbf{n}$ induce a normalization that results in $\hat{\mathbf{W}}$ being approximately bistochastic (we use the normalization of [Mazumdar et al. 2017]). With this factorization we can perform a variable substitution from pixel-space into "bilateral-space":

$$\mathbf{x} = \mathbf{S}^{\mathrm{T}}\mathbf{y} \tag{10}$$

where $\mathbf{y}$ contains values for each bilateral grid vertex and $\mathbf{x}$ contains values for each pixel. Assuming the σ_* parameters are not small, $\mathbf{y}$ will be substantially smaller than $\mathbf{x}$. We turn the expensive pixel-space optimization problem in Equation 7 into a tractable bilateral-space optimization problem:

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} - \mathbf{b}^{\mathrm{T}}\mathbf{y} + c \\ \mathbf{A} &= \lambda(\operatorname{diag}(\mathbf{m}) - \operatorname{diag}(\mathbf{n})\mathbf{B}\operatorname{diag}(\mathbf{n})) + \operatorname{diag}(\mathbf{S}\mathbf{c}) \\ \mathbf{b} &= \mathbf{S}(\mathbf{c} \circ \mathbf{t}) \qquad c = \frac{1}{2}(\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}\mathbf{t} \end{aligned} \tag{11}$$

where $\mathbf{y}$ is the solution to the problem in bilateral-space, and $\circ$ denotes a Hadamard product. Solving this problem simply requires solving a sparse linear system and undoing our variable substitution:

$$\hat{\mathbf{x}} = \mathbf{S}^{\mathrm{T}}(\mathbf{A}^{-1}\mathbf{b}) \tag{12}$$

Following [Mazumdar et al. 2017], we solve this sparse linear system using preconditioned heavy ball optimization, which produces similar results to the preconditioned conjugate gradient used in [Barron and Poole 2016] while being better suited to a fast implementation.

The bilateral solver as previously described is capable of producing edge-aware smooth depth maps from noisy or incomplete inputs and can be made to produce real-time results running on a mobile CPU. In the following sections we describe a number of improvements to the solver. First, we generalize the solver (or indeed any linear smoothing operator) to produce output that is smooth in a *coplanar* sense, rather than smooth in a fronto-parallel sense. This formulation results in significantly improved output on scenes containing foreshortened planes (walls, floors, etc.), which are common in photographic and AR contexts. Second, we present a simple and cheap method for inducing real-time temporal consistency in the solver, as well as an approach for "warm starting" multiple instances of a solver that improves convergence rates in a real-time / video-processing context. Third, we demonstrate that the solver can be used for "late stage" slicing, in which we use a bilateral grid computed from earlier stereo inputs to produce edge-aware depth maps from the most recent viewfinder frame. The result is extremely low-latency output that is still edge-aware.

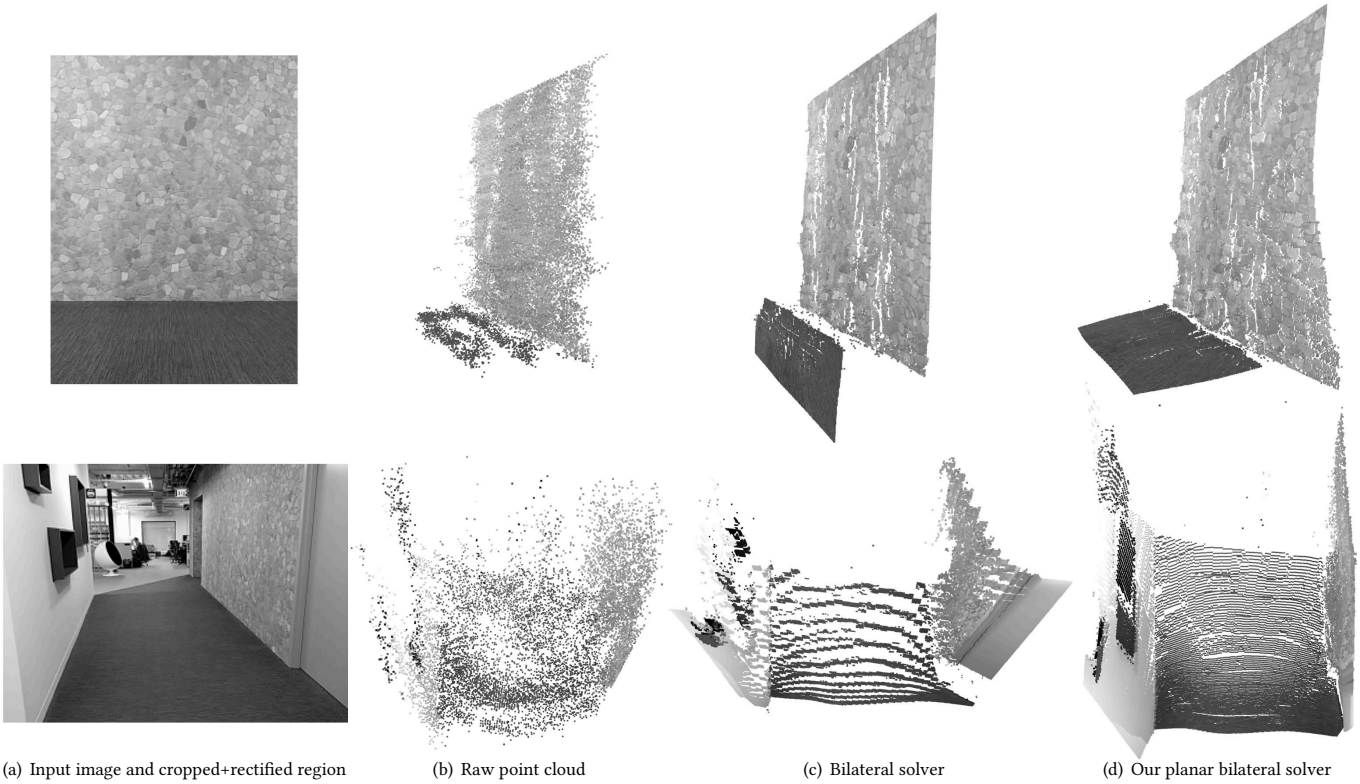
7.1 Planar Bilateral Solver

The bilateral solver produces a per-pixel labeling that, along with a data term, minimizes the squared distances between pixels that are spatially nearby and have similar colors or grayscale intensities. The output of the solver (ignoring the data term) is therefore an image containing a single constant value, which in a stereo context means that the solver is strongly biased towards producing fronto-parallel depth maps. This bias is problematic, as real-world environments frequently contain surfaces that are smooth or flat but *not* fronto-parallel, such as floors, walls, and ceilings. For an illustration of how this can be problematic, see Figure 8, which shows that the regular bilateral solver's output may be dramatically incorrect in the presence of flat but not fronto-parallel surfaces, causing foreshortened surfaces to be erroneously recovered as "billboard"-like flat surfaces oriented perpendicularly to the camera.

This fronto-parallel bias in depth estimation has been noted in the literature. [Woodford et al. 2009] and [Zhang et al. 2014] address this issue using specialized optimization algorithms designed to recover depth maps that minimize second-order variation, rather than first-order variation. These approaches work well, but are too expensive for real-time use and do not appear to be amenable to fast bilateral-space optimization. [Furukawa et al. 2009] present a stereo pipeline built around the assumption that the world is "Manhattan-like", but this rigid assumption hurts performance in the many cases that do not obey this strong assumption of co-planarity, such as human subjects or the natural (not man-made) world. The trilateral filter of [Choudhury and Tumblin 2005] modifies a bilateral filter to produce piece-wise linear output, but this approach does not provide a way to similarly modify the bilateral solver.

Our approach is to simply embed the bilateral solver in a per-pixel plane-fitting algorithm, such that the minimal-first-order-variation assumption of the bilateral solver causes the final output of our algorithm to have low second-order variation. Local plane-fitting is well studied in the literature [Klasing et al. 2009; Wang et al. 2001], usually under the assumption that the spatial support of each plane fit is limited. By using the bilateral solver as the engine for aggregating plane-fit information, we can perform global and edge-aware plane-fitting. This causes our recovered depth map to not be confounded by foreshortened surfaces, as it can explain away such surfaces as simply being slanted planes.

Our approach, which we dub the "planar bilateral solver", fits a plane to each pixel in the image in a moving least squares context, where the interpolator in each pixel's least-squares fit is the output of a bilateral solver. Implicitly we construct a 3D linear system at every pixel, in which the left and right hand sides of each linear system use the bilateral solver to compute the "scatter" matrix used in standard plane fitting. This approach is in no way dependent on any property of the bilateral solver except that the solver is a linear filter whose weights are always non-negative, and so we describe this plane-fitting procedure in completely general terms. Because the math for plane-fitting is tedious and well-understood, for the sake of brevity and reproducibility we describe this "planar filtering" approach primarily using pseudo-code, in Algorithm 1. This planar_filter(·) operator takes as input some image Z to be filtered (in our case a depth map), some non-negative filtering operator filter(·) (in our



(a) Input image and cropped+rectified region (b) Raw point cloud (c) Bilateral solver (d) Our planar bilateral solver

Fig. 8. Here we demonstrate the efficacy of our planar bilateral solver. In (a) we have two input images from our pipeline showing common indoor environments. In (b) we have a raw point cloud from stereo matching, visualized from a different angle than the camera. In (c) we have the output of the standard bilateral solver, which produces reasonable results when surfaces happen to be fronto-parallel (ie, on the wall in the top image) but exhibits significant artifacts when this fronto-parallel bias is incorrect. In (d) we see that our planar bilateral solver resolves these issues and produces significantly improved output, in which the heavily foreshortened walls and floor are more accurately recovered.

case a bilateral solver), and some regularization parameter ϵ. This algorithm filters the outer product of $(x, y, Z, \mathbf{1})$ with itself, which gives us the left- and right-hand sides of a linear system at each pixel that we represent as a 6-channel image A and 3-channel image b (x and y are the coordinates of each pixel, $\mathbf{1}$ is an image of all 1's, $\circ$ is the Hadamard product, and / is element-wise division). With these images we can solve the linear system at each pixel in parallel using an LDL decomposition (which we found to be more stable than other options we explored). Pseudo-code for this per-pixel solve can be found in Algorithm 2. In our pseudo-code we recover Z_z, the constant offset of each pixel's plane fit, which we use as the output depth at each pixel. Our pseudo-code also computes (Z_x, Z_y), the gradient of Z_z at each pixel, which we do not use in this work.

The parameter ϵ biases the output of planar filtering to be fronto-parallel, by imposing Tikhonov regularization on the gradient of the recovered surface. As ϵ approaches ∞ the output of planar_filter$(\cdot)$ exactly approaches the output of filter$(\cdot)$. The standard bilateral solver can therefore be viewed as a planar bilateral solver that has been heavily regularized to produce maximally fronto-parallel output — further demonstrating the standard bilateral solver's fronto-parallel bias. We set $\epsilon = 1$ in our experiments.

Because a planar filter requires calling filter$(\cdot)$ 9 times, and because calling this filter is significantly more expensive than performing element-wise per-pixel math, applying a planar bilateral solver is roughly 9× more expensive than applying a standard bilateral solver. This can be reduced somewhat by noting that filter$(\mathbf{1}) = \mathbf{1}$, and by calculating the other 8 bilateral solver instances in parallel which allows for easy vectorization. The next section will demonstrate that, in practice, the planar solver benefits substantially from temporal warm-start initialization, thereby giving it runtimes that are not much more than those of the regular bilateral solver.

7.2 Temporal Smoothness, Warm-Start Initialization, and Late-Stage Slicing

To produce a compelling user experience, the depth maps produced by our system must be smooth over time. This goal of temporal consistency is somewhat at odds with our need for responsive, low-latency output that is tightly aligned to the edges of the current viewfinder frame. In this section, we present a simple and effective method for the bilateral solver and planar bilateral solver to produce high quality, temporally consistent real-time results.

Algorithm 1 planar_filter(Z, filter($\cdot$), ϵ)
Input: Some image to be filtered Z, some linear averaging filter filter($\cdot$), and some non-negative regularization parameter ϵ.
Output: A three channel image of per-pixel plane fits (Z_x, Z_y, Z_z).

$[F_1, F_x, F_y, F_z, F_{xx}, F_{xy}, F_{xz}, F_{yy}, F_{yz}]$
$\quad \leftarrow \text{map}(\text{filter}(\cdot), [\mathbf{1}, x, y, Z, x \circ x, x \circ y, x \circ Z, y \circ y, y \circ Z])$
$A_{1,1} \leftarrow F_1 \circ y \circ y - 2(y \circ F_y) + F_{yy} + \epsilon^2$
$A_{1,2} \leftarrow F_1 \circ y \circ x - x \circ F_y - y \circ F_x + F_{xy}$
$A_{2,2} \leftarrow F_1 \circ x \circ x - 2(x \circ F_x) + F_{xx} + \epsilon^2$
$A_{1,3} \leftarrow F_1 \circ y - F_y$
$A_{2,3} \leftarrow F_1 \circ x - F_x$
$A_{3,3} \leftarrow F_1 + \epsilon^2$
$b_1 \leftarrow F_z \circ y - F_{yz}$
$b_2 \leftarrow F_z \circ x - F_{xz}$
$b_3 \leftarrow F_z$
$(Z_x, Z_y, Z_z) \leftarrow \text{solve_image_ldl3}(A, b)$

Algorithm 2 solve_image_ldl3(A, b)
Input: A 6-channel image A and a 3-channel image b, where channels in A correspond to the upper triangular part of a 3×3 matrix.
Output: A three-channel image x where for each pixel i in the input linear system, $x(i) = A(i) \backslash b(i)$ using an LDL decomposition.

$d_1 = A_{1,1}$
$L_{1,2} = A_{1,2}/d_1$
$d_2 = A_{2,2} - L_{1,2} \circ A_{1,2}$
$L_{1,3} = A_{1,3}/d_1$
$L_{2,3} = (A_{2,3} - L_{1,3} \circ A_{1,2})/d_2$
$d_3 = A_{3,3} - L_{1,3} \circ A_{1,3} - L_{2,3} \circ L_{2,3} \circ d_2$
$y_1 = b_1$
$y_2 = b_2 - L_{1,2} \circ y_1$
$y_3 = b_3 - L_{1,3} \circ y_1 - L_{2,3} \circ y_2$
$x_3 = y_3/d_3$
$x_2 = y_2/d_2 - L_{2,3} \circ x_3$
$x_1 = y_1/d_1 - L_{1,2} \circ x_2 - L_{1,3} \circ x_3$

Temporally consistency using a bilateral solver has been previously accomplished in [Anderson et al. 2016], who appended an extra "temporal" dimension to a bilateral solver and solved for the per-pixel depth labeling for an entire video sequence using one instance of the solver. This method produces high-quality results, but is not applicable to our real-time use-case where frames must be processed as they are acquired. We therefore use a causal IIR-like approach to temporal smoothness, in which we track a single bilateral grid of estimated depths, and repeatedly update this bilateral grid using the output of a single-image bilateral solver instance on each incoming frame. This approach can run in real-time, and allows us to use "late stage" slicing to produce extremely low latency edge-aware output — a critical feature in augmented reality applications.

As reviewed in Section 7, the baseline single-image bilateral solver estimates a depth map by implicitly constructing and solving a linear system $\mathbf{A}^{-1}\mathbf{b}$ in bilateral space, and then using a "slice" matrix $\mathbf{S}^{\mathrm{T}}$ to produce a per-pixel labeling $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} \leftarrow \mathbf{S}^{\mathrm{T}}(\mathbf{A}^{-1}\mathbf{b}) \tag{13}$$

In our temporally consistent solution, we track an exponential moving average of a bilateral grid of depths $\bar{\mathbf{y}}$, which we initialize to $\mathbf{0}$. For each input image t, we solve for the current frame's bilateral grid of depths $\hat{\mathbf{y}}_t$ by solving a linear system $\mathbf{A}_t^{-1}\mathbf{b}_t$, and then update $\bar{\mathbf{y}}$ using exponential decay. We slice from that averaged bilateral grid to produce a per-pixel depth estimate. The exact update applied at frame t is:

$$\begin{aligned} \hat{\mathbf{y}}_t &\leftarrow \mathbf{A}_t^{-1}\mathbf{b}_t \\ \bar{\mathbf{y}} &\leftarrow \alpha \,\text{blur}\,(\bar{\mathbf{y}}) + (1-\alpha)\hat{\mathbf{y}}_t \\ \hat{\mathbf{x}}_t &\leftarrow \mathbf{S}_t^{\mathrm{T}}\left(\frac{\bar{\mathbf{y}}}{1-\alpha^t}\right) \end{aligned} \tag{14}$$

where α is a parameter that controls how much temporal smoothness is encouraged, blur($\cdot$) applies a normalized $[1, 4, 6, 4, 1]$ blur along the three dimensions of the bilateral grid of $\bar{y}$, and the division by $(1 - \alpha^t)$ serves to "unbias" our moving average estimate of $\bar{y}$. The blur($\cdot$) operator diffuses information spatially between frames, which appears to help, and the linear interpolation according to α diffuses that same information temporally.

Tracking a small bilateral grid of depths instead of a large per-pixel depth map has an obvious speed advantage, as each frame's update requires significantly less compute. However, tracking and blurring a bilateral grid of depths in this manner instead of tracking a depth map also means that our temporal smoothing method is invariant to small camera or scene motion, without the need to explicitly estimate per-pixel motion between frames. Similarly, tracking depth in bilateral-space allows for us to produce extremely low-latency edge-aware depth estimates through a process we call "late-stage slicing": after updating $\bar{\mathbf{y}}$, if the viewfinder frame has changed we use the new frame to construct $\mathbf{S}_t^{\mathrm{T}}$ and slice out a per-pixel depth labeling. Since slicing is significantly faster than solving, this approach lets us produce per-pixel depth maps that are exactly aligned to the current viewfinder frame, thereby enabling compelling and responsive visual effects. This late-stage slicing approach also allows our algorithm to degrade gracefully on low end mobile devices where our per-frame depth estimation may take longer than the 17 or 33 milliseconds between frames. This method is similar in spirit to the "time warp" approach used in some VR applications [Van Waveren 2016], though our technique uses just raw pixel intensities instead of geometric tracking information.

Our temporal smoothing approach enables another acceleration, in which we "warm start" each frame's bilateral solver instance by initializing the bilateral grid being solved for via gradient descent using the previous frame's solution. Since adjacent frames have similar image content, except in the presence of extreme motion, so this approach significantly improves convergence and allows for fewer gradient descent iterations in all but the first frame, as we demonstrate in Section 9.3.

Our temporal consistency approach generalizes straightforwardly from the regular bilateral solver to our planar bilateral solver. We apply temporal consistency to the linear systems describing the per-pixel plane fits before the LDL solve step, which produces better results than applying a temporal filter to the estimated depth at each

frame. Our late-stage slicing also includes a late-stage per-pixel LDL solve, which increases runtimes slightly, but not enough to preclude real-time performance.

8 EVALUATION

In this section we perform exhaustive evaluations of the entire pipeline. We stress the system in various scenarios, reporting quantitative comparisons on standard benchmarks as well as qualitative comparisons with state of the art methods. We start the evaluation by testing our framework on the Middlebury Stereo Dataset V3 [Scharstein et al. 2014], showing that our sparse matcher is among the top performers on the benchmarks and the full system is on par with fast real-time stereo matching approaches. We then compare with deep learning based solutions on challenging scenes and finally show how the method performs favorably compared to the state of the art in the mobile industry (iPhoneX).

8.1 Stereo-Matcher Evaluation on Middlebury Dataset

We here quantitatively compare our stereo matching approach to other local state-of-the-art algorithms on the Middlebury Stereo Dataset V3 [Scharstein et al. 2014]. We submitted the results on the Middlebury benchmark[1]. Our proposed sparse stereo matching with invalidation step (Section 6.2) is among the top performers on the sparse benchmark for multiple reported metrics. The output of the sparse matching is indeed accurate and could be potentially used for reconstruction tasks.

On the dense benchmark, the proposed method performs very well in comparison to other fast methods such as the Intel R200 algorithm that requires a custom ASIC. Given computational constraints available on mobile platforms, it is expected that offline techniques such as [Taniai et al. 2018] are able to provide more accurate results. However, we want to point out that the goal of the densification step is to produce edge aligned depth maps for occlusion handling rather than minimizing average depth error. This means that, for the task at hand, we can tolerate some imprecision in the predicted depth, but we are sensitive to small edge misalignment. Therefore, these standard metrics are not well suited to evaluate what our algorithm is designed for.

In Table 1 we report the results using the *bad 2.0* error, which is the standard metric for this dataset and computes the percentage of pixels with error greater than 2 disparities. As representative competitors, we picked the Intel R200 algorithm, which is based also on binary descriptors followed by a more sophisticated optimization scheme that uses Semi-Global Matching (SGM) [Hirschmuller 2008]; and the current state-of-the-art on this dataset, LocalExp [Taniai et al. 2018], which uses a combination of deep features together with a PatchMatch-like scheme with 3D slanted support windows. As shown in the table, our method beats the Intel R200 and as expected is far from the state-of-the-art given the strict computational resources available on mobile platforms.

Table 1. Quantitative results (bad 2.0) on Middlebury Dataset V3 (dense). It is interesting to note that we get slightly better results than the fast Intel R200 method which uses semi-global matching. The state-of-the-art, LocalExp [Taniai et al. 2018], requires computational resources that are significantly exceeding what is available on mobile platforms.

	Austr	AustrP	Bicyc2	Class	ClassE	Compu	Crusa	CrusaP	Djemb	DjembL	Hoops	Livgrm	Nkuba	Plants	Stairs	Average
Intel R200	70.5	14.4	21.3	37.7	72.2	38.1	53.2	31.4	18.3	52.4	52.6	44.1	45.4	50.7	66.5	40.9
LocalExp	3.65	2.87	2.98	1.99	5.59	3.37	3.48	3.35	2.05	10.3	9.75	8.57	14.4	5.4	9.55	5.43
Ours	67.6	25.0	29.2	40.9	57.3	35.5	57.5	40.4	19.9	42.8	52.6	39.8	37.1	51.7	34.9	40.4

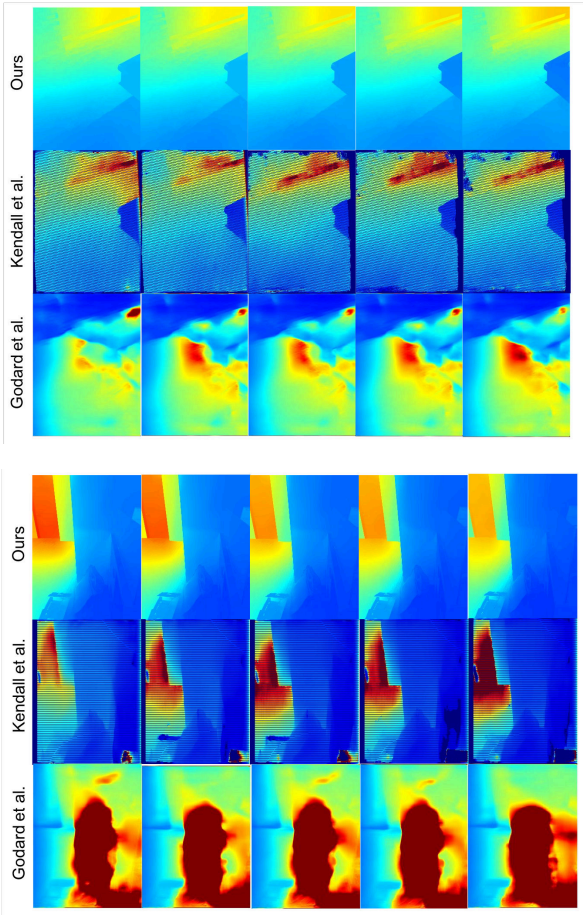


Fig. 9. Comparison to deep neural networks. Even computationally expensive networks fail to produce a valid temporally consistent depth map of the scene. The monocular method of [Godard et al. 2017], for which they offer a version fine-tuned on indoor scenes from the NYU Depth dataset [Silberman et al. 2012]. The line artifacts in [Kendall et al. 2017] are due to the re-projection of disparity images to the un-rectified image. Note that both baselines have not been fine-tuned on our data, hence we present them here for illustration purposes.

8.2 Qualitative Comparisons with Deep Learning solutions

In this section we compare the method with deep learning based state-of-the-art solutions. Recent work relies on complex deep architectures to infer depth from single images [Eigen et al. 2014; Godard et al. 2017; Laina et al. 2016] as well as stereo configurations [Kendall et al. 2017; Khamis et al. 2018; Pang et al. 2017]. In Figure 9 we show the comparisons of our method with two recent methods for monocular [Godard et al. 2017] and stereo [Kendall et al. 2017], respectively. Notice that the monocular method of [Godard et al. 2017] is fine-tuned on indoor scenes from the NYU Depth dataset [Silberman et al. 2012]. Despite this, it suffers from gross errors on these challenging scenes. The two-view method of [Kendall et al. 2017] is instead more robust and produces reasonable results even in textureless regions. However both the competitors fail to produce temporally consistent depth maps of the scene. In contrast, our algorithm is more robust to temporal changes and does not completely fail in textureless areas. We want to point out that our RGB data is different (e.g. resolution) compared to the RGB data these baselines have been trained with. Fine-tuning these baselines on that data could lead to some quality improvement, but is unlikely to address gross errors and the temporal instability of predictions.

8.3 Qualitative Comparisons with the iPhoneX

The iPhoneX is the latest generation of Apple smartphones. On the back, the iPhoneX possesses two cameras that have different focal length. Using this stereo pair, the iPhoneX computes 320×240 depth maps at 24Hz. In contrast to a wide range of mid-level smartphones iPhoneX has access to far more powerful computational resources. Given that the iPhoneX can leverage a rigid camera setup and at least one zoomed camera, one would expect their results to be more precise and more stable than ours. In general, the iPhoneX provides good results on well-texture environments, but does not manage to reasonably handle harder cases as can be observed in Figure 10. Side by side video comparisons are available in the supplementary materials, in which one can observe that the proposed system provides depth maps that are much more temporally stable.

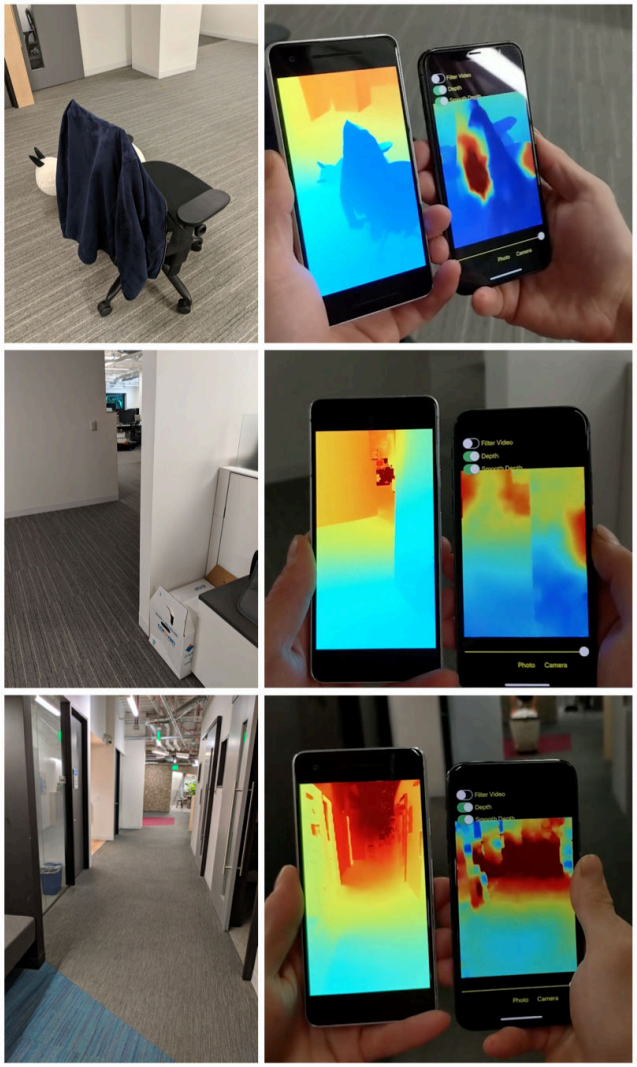


Fig. 10. Qualitative comparison with the iPhoneX. For these experiments, we placed the iPhoneX and a Pixel 2 closely together with the iPhoneX on the right, and the Pixel 2 equipped with the proposed solution on the left. The iPhoneX struggles at providing useful data in low-textured regions. Note that the Pixel 2 is less powerful compared to the iPhoneX.

9 ABLATION STUDY

In this section, we evaluate the contribution of each component of the pipeline through quantitative and qualitative experiments.

9.1 Invalidation Experiments

To evaluate the accuracy of the invalidation reached by the proposed machine learning solution, we acquired 10000 stereo frames of indoor scenes: we use 8000 frames for training and 2000 frames for testing. For each frame, we produce sparse disparity maps as described in Section 6 and perform an initial invalidation using the CRF cost as detailed in Section 6.2. We then generate 'ground-truth', outlier-free disparity maps, running an offline connected component analysis that removes regions smaller than 300 pixels. Examples of input and generated groundtruth pairs can be observed in Figure 7. We then train a decision tree using the procedure described in Section 6.2. The proposed machine learning based invalidation scheme reaches an accuracy of 93% for a tree of depth 7 compared to the 'ground-truth' generated by a connected component analysis. As shown in Figure 11, the accuracy rapidly improves until a tree depth of 7, after which the gains become marginal. At this tree depth, the proposed solution is 85% faster than the connected component analysis. We note that the main differences between connected component invalidation, and tree-based invalidation, appear when the size of regions fall right below the hand-defined threshold (300), as illustrated in Figure 12.

[1]See "MotionStereo" entry on the official dataset website.

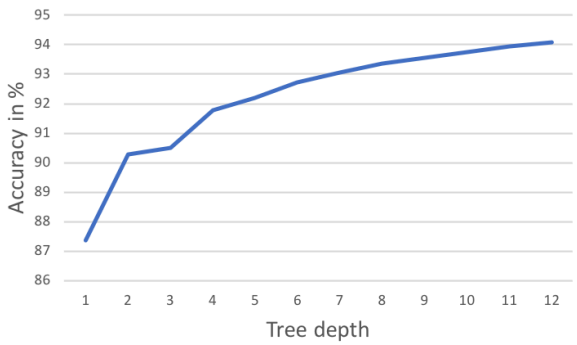


Fig. 11. Accuracy of the proposed machine learning invalidation. Notice how the accuracy rapidly increases up until a tree depth of 7, after which each level of the tree only marginally improves results.

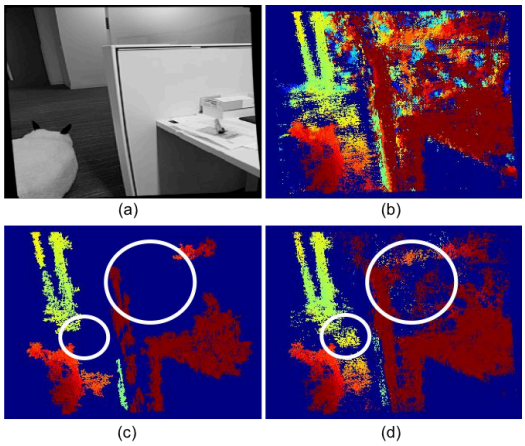


Fig. 12. Largest difference (14%) between connected component based invalidation and the proposed tree-based invalidation in the test set. (a) rectified image, (b) disparity after crf-cost invalidation, (c) connected component based invalidation, (d) tree-based invalidation.

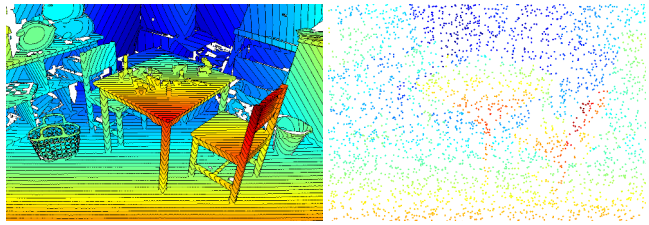
9.2 Planar Bilateral Solver Experiments

The standard bilateral solver can be thought of as 'averaging' depth within cells of the grid leading to a fronto-parallel bias. This behaviour is particularly exacerbated in cases like in the top row of Figure 8, where the input point cloud offers a decent initialization, but where the bilateral solver provides a prediction for the floor that is off by almost 90°. Although the input point-cloud might be noisy, the proposed planar bilateral solver estimates solutions that are much closer to reality than those obtained by the bilateral solver.

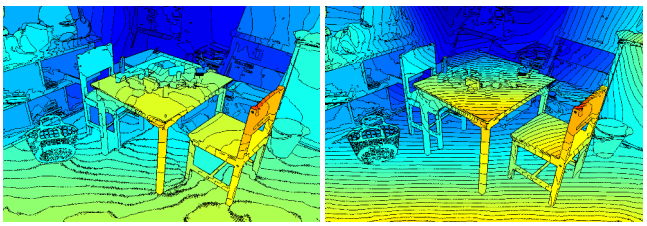
To quantitatively evaluate the planar bilateral solver we present an experiment using the Middlebury Stereo Dataset V3 [Scharstein et al. 2014]. We simulated our own stereo task by randomly deleting 99.75% of each ground-truth depthmap's pixels by setting their values and confidences to zero, and then using these sparse depths as input to a (planar) bilateral solver. Using the half-resolution training images from the Middlebury dataset, we found the optimal parameters for the baseline bilateral solver (our own implementation which produces nearly identical performance to that of [Mazumdar et al. 2017]) by minimizing the geometric mean of the RMSEs for the 10 images ($\lambda = 0.5$, $\sigma_{xy} = 8$, $\sigma_r = 4$). We then evaluated this task using a planar bilateral solver, with the same parameters as the baseline solver and with the planar solver's one additional parameter set to $\epsilon = 0.1$ (performance was not sensitive to this value). Results for this task can be seen in Table 2. On scenes containing slanted ground planes and walls (e.g. 'Piano', 'Playtable') the planar solver gives a significant improvement of 7-15%. Scenes consisting of mostly fronto-parallel surfaces (e.g. 'Shelves', 'Recycling') yield no improvement as the baseline bilateral solver's assumptions are satisfied, but do not exhibit any significant loss in quality. A visualization of the output of both solvers with respect to ground truth for a scene containing slanted surfaces can be seen in Figure 13. As illustrated in Figure 14, it is interesting to see that even for scenes where planar assumptions are violated, the proposed solver is able to recover solutions that can be superior to the baseline solver.

Table 2. Here we report RMSEs of our planar bilateral solver and the standard bilateral solver on a depth-inpainting task using the Middlebury Stereo Dataset V3. On scenes containing foreshortened floors and walls the planar solver produces a 7-15% reduction in RMSE, while performance on scenes consisting of mostly fronto-parallel surfaces is largely unaffected.

	Adirondack	Jadeplant	Motorcycle	Piano	Pipes	Playroom	Playtable	Recycle	Shelves	Vintage	Geo. Mean
Baseline Solver [2017]	11.48	35.97	16.05	9.09	16.83	13.10	10.06	8.01	8.13	16.70	13.07
Planar Bilateral Solver	11.14	34.92	15.59	8.38	16.41	12.67	8.54	8.03	8.15	14.46	12.39
Relative Improvement	+2.9%	+2.9%	+2.8%	+7.8%	+2.5%	+3.3%	+15.1%	-0.2%	-0.3%	+13.4%	+5.2%

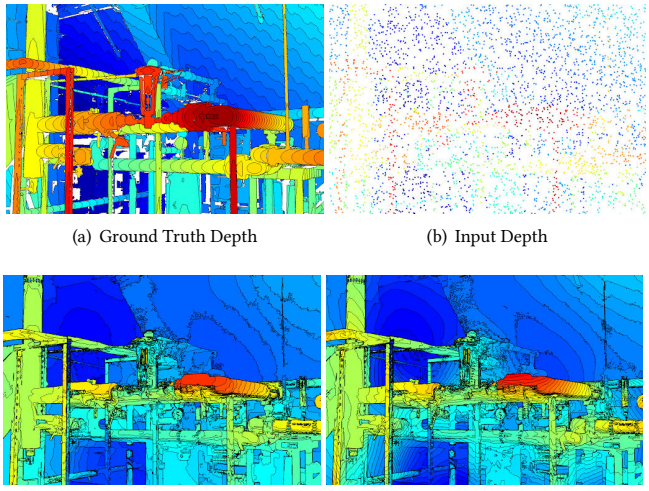


(a) Ground Truth Depth (b) Input Depth



(c) Baseline Solver [2017] Output (d) Planar Bilateral Solver Output

Fig. 13. A visualization of our depth-inpainting task for evaluating the planar bilateral solver. A ground-truth depth map (a) is randomly decimated to produce an input depth map (b) that is used as input to the standard bilateral solver (c) and our planar bilateral solver (d). We see that the planar solver recovers the foreshortened surfaces on the walls, floor, table, and chair, while the baseline solver produces oversmoothed and fronto-parallel surfaces. Depths in (a, c, d) are rendered as filled contour plots to better emphasize gradients and level sets, and (b) is rendered by dilating the sparse input depths for better visibility.



(a) Ground Truth Depth (b) Input Depth

(c) Baseline Solver [2017] Output (d) Planar Bilateral Solver Output

Fig. 14. On scenes that violate the planar assumption of the planar bilateral solver, the planar solver's output closely resemble that of the baseline solver.

9.3 Temporal Consistency Evaluation

The proposed temporal filtering approach leads to results that are significantly more temporally stable. Figure 15 shows that although the raw depth maps might vary from one frame to the next, our approach is capable of producing temporally stable results. Our temporal consistency approach is very fast, using approximately 0.13% of the total computation of the bilateral solver. We also compare to computationally expensive deep neural network based approaches that give temporally less stable results or suffer from gross errors.

In Figure 16, we evaluate the proposed warm-start strategy presented in Section 7.2. We compare warm-start initialization against a baseline all-zero initialization and the heuristic initialization of [Mazumdar et al. 2017], which our warm-start technique also uses on the first frame. Not only does warm-start outperform the heuristic initialization, but it requires no additional computation, thereby letting us save 6.6% of total runtime of the solver that was previously spent on heuristic initialization. Figure 16 shows 64 iterations of gradient descent per frame for illustration's sake. In practice, we use 25 iterations per frame, which (when matching the final loss of the solver) gives a 1.8 times speed-up compared to the baseline solver and a 1.45 times speed-up compared to the initialization of [Mazumdar et al. 2017], as illustrated in Figure 16 (a). The improvement is even more pronounced for the planar solver, as depicted in Figure 16 (b).

9.4 Late-stage Slicing Evaluation

In Figure 17, we show the impact of the late-stage slicing on the resulting occlusions. Without the late-stage slicing the latency in the depth computation becomes evident as the user moves the phone, resulting in the occlusion boundary not aligning with the physical foreground object boundary. In contrast, our late-stage slicing mitigates this latency and produces visually pleasing occlusions where the occlusion boundary correctly hugs the foreground object.

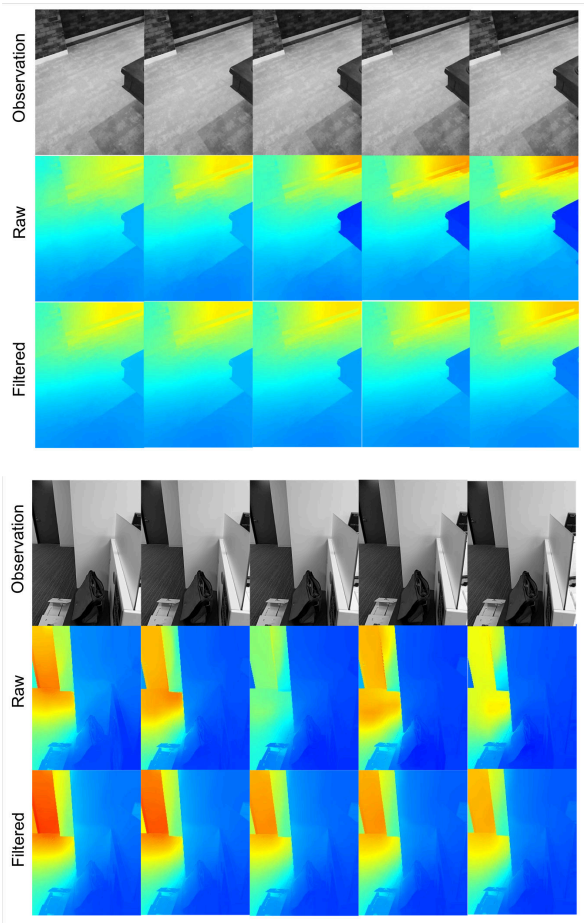
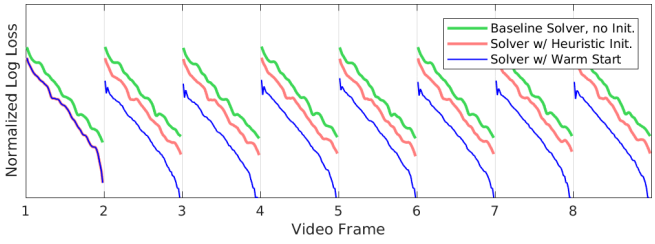


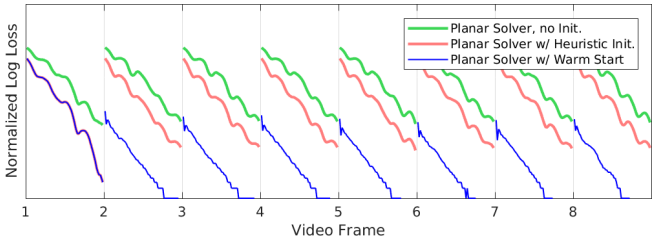
Fig. 15. Temporal filtering significantly improves the stability of the produced results.

10 AR APPLICATIONS

In this section we demonstrate that the proposed system can effectively enable occlusions for applications such as AR shopping, AR photos as well as AR navigation. To this end, we blend a virtual AR Asset, e.g. a virtual chair, into the RGB feed of the smartphone using the computed depth map. In more detail, we create the composite image C by applying a convex combination of the image I coming from the camera feed and the virtual asset image A (created by rendering the virtual asset from the current viewpoint). Naturally, the weight associated to I can be set to 1 if the depth of the virtual asset is greater then the computed depth map, and 0 otherwise. However, we found that results are more visually pleasing when computing the weight of A based on the distance of the virtual object to the computed depth map as $1/(1 + e^{-0.5 \cdot (D - D_0)})$, where D and D_0 are the computed and virtual depth values respectively.



(a) Bilateral Solver



(b) Planar Bilateral Solver

Fig. 16. The (normalized, log) loss of a bilateral solver (a) and a planar bilateral solver (b) on an 8-frame video sequence, with and without warm-starting. Compared to the heuristic initialization of [Mazumdar et al. 2017], initializing each frame's solver to the previous solution hastens convergence, particularly in the case of the planar solver.

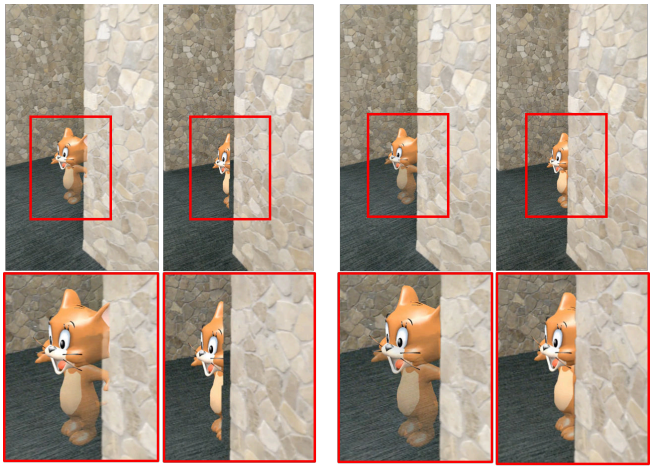


Fig. 17. Impact of late-stage slicing. Left: without late-stage slicing the latency in the depth computation becomes visible because the occlusion boundary is not aligned with the foreground object. Right: with our late-stage slicing the latency is mitigated and the occlusion boundary nicely hugs the object boundary.

This results in the object gradually fading in and out when temporal depth inconsistencies are present.

Figure 20 shows qualitative results for a variety of challenging indoor and outdoor scenes. Note that our approach generates visually appealing results even for challenging occluders such as untextured walls or complex shaped plants.

Table 3. Average timings (in ms) of the different pipeline components on mid and high end phones. These numbers were obtained by averaging run-times over a few hundred calls while running the whole system on real data.

Stage	Component	A7	Pixel XL	S7	S8	Pixel 2
Dense depth map	Keyframe selection	2.6	2.3	2.0	1.7	1.3
	Rectification	79.7	33.8	38.0	39.5	31.7
	CRF	60.0	23.7	27.1	26.2	27.2
	Invalidation	13.4	5.6	6.2	4.5	4.5
	Triangulation	16.3	5.6	8.5	4.8	4.7
	Bilateral solver	42.3	12.9	24.6	16.9	16.5
	Planar solver	71.6	31.1	48.6	32.15	30.9
On-demand rendering	Late-stage slicing	5.0	1.7	2.7	2.1	1.8
	Rendering	0.3	0.4	0.4	0.2	0.4

10.1 Run-time breakdown

Table 3 shows the timings of the different system components for a variety of different mid to high end phones. Note that we either use the planar or the bilateral solver but not both. It is important to note that although the run-time of components required for dense depth estimation sum up to more than 33ms, the system latency is under 6ms on all devices. This latency corresponds to the run-time of the late-stage slicing and rendering, which run independently to depth estimation pipeline. This guarantees a smooth user experience at 30Hz across all tested smart-phones.

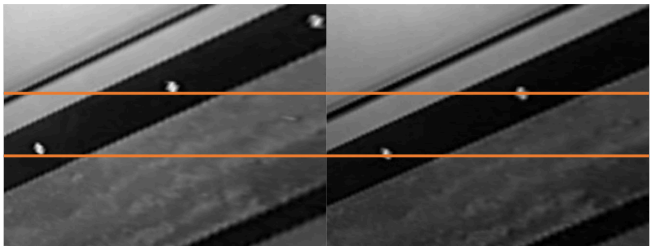


Fig. 18. Left and right rectified image with an imprecise relative pose. In this example, the correspondences are 14 lines apart.

11 LIMITATIONS

The proposed system suffers from the usual limitations of monocular depth estimation systems. A lot of components can be impacted when the relative pose between the current frame and the selected keyframe is imprecise. For instance, as illustrated in Figure 18, pose imprecision can lead to correspondences that are several lines apart, significantly increasing the difficulty of stereo matching. Robust stereo matching can sometimes still be performed although camera poses are imprecise, but then triangulation leads to results that exhibits visible scale change as can be observed in Figure 19. Another limitation comes from hardware constraints that can for instance manifest in the form of rolling-shutter artifact and motion blur. Finally, as it is well known in the passive-stereo literature, low-textured areas are particularly ambiguous and performing inference over them often lead to incorrect results.

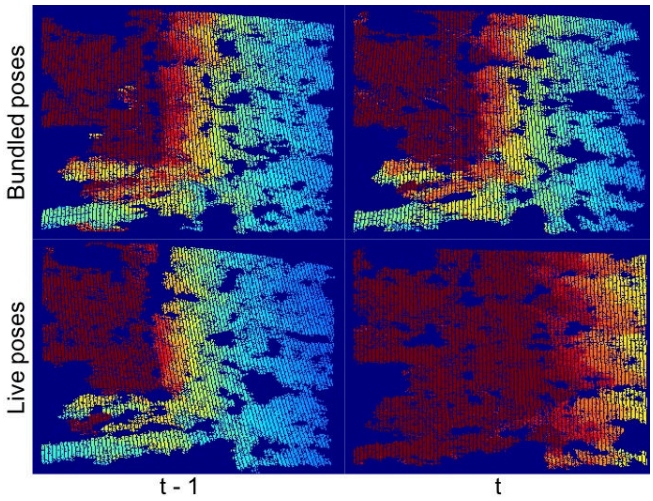


Fig. 19. Pose imprecision can lead to sudden scale change. The top row shows how the sparse depth evolves between two successive time stamps. The bottom row depicts how the imprecise live poses can lead to rapid scale changes between time $t-1$ and time t despite disparities being predicted correctly.

12 CONCLUSIONS

In this article we have presented several technical contributions that for the first time allow predicting low-latency, edge aligned, and dense depth-maps on a single CPU core of a mobile phone. We have demonstrated that this method is effective in occlusion handling, creating a plausible illusion that real and virtual objects exist in the same environments. Directions for future work include using deep-reinforcement learning for directly predicting occlusion masks and working on reducing the computational requirements of such architectures.

REFERENCES

Robert Anderson, David Gallup, Jonathan T Barron, Janne Kontkanen, Noah Snavely, Carlos Hernández, Sameer Agarwal, and Steven M Seitz. 2016. Jump: Virtual Reality Video. *SIGGRAPH Asia* 35, 6 (2016), 198.

Apple. 2018. ARKit | Apple Developer Documentation. (2018). https://developer.apple.com/documentation/arkit

Christian Bailer, Manuel Finckh, and Hendrik PA Lensch. 2012. Scale robust multi view stereo. In *European Conference on Computer Vision*. Springer, 398–411.

Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. 2009. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM TOG* 28, 24 (2009).

Jonathan T Barron, Andrew Adams, YiChang Shih, and Carlos Hernández. 2015. Fast Bilateral-Space Stereo for Synthetic Defocus. *CVPR* (2015).

Jonathan T Barron and Ben Poole. 2016. The Fast Bilateral Solver. *ECCV* (2016).

Michael Bleyer, Christoph Rhemann, and Carsten Rother. 2011. PatchMatch Stereo-Stereo Matching with Slanted Support Windows. *BMVC* (2011).

Yuri Boykov, Olga Veksler, and Ramin Zabih. 2001. Fast approximate energy minimization via graph cuts. *TPAMI* (2001).

Gary Bradski and Adrian Kaehler. 2000. OpenCV. *Dr. Dobbs journal of software tools* 3 (2000).

Prasun Choudhury and Jack Tumblin. 2005. The trilateral filter for high contrast images and meshes. *SIGGRAPH Courses* (2005).

Antonio Criminisi, Jamie Shotton, Ender Konukoglu, et al. 2012. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision* (2012).

David Eigen, Christian Puhrsch, and Rob Fergus. 2014. Depth map prediction from a single image using a multi-scale deep network. *NIPS* (2014).

Daniel Evangelakos and Michael Mara. 2016. Extended TimeWarp latency compensation for virtual reality. *Interactive 3D Graphics and Games* (2016).

Sean Ryan Fanello, Julien Valentin, Adarsh Kowdle, Christoph Rhemann, Vladimir Tankovich, Carlo Ciliberto, Philip Davidson, and Shahram Izadi. 2017a. Low Compute and Fully Parallel Computer Vision with HashMatch. *ICCV* (2017).

Sean Ryan Fanello, Julien Valentin, Christoph Rhemann, Adarsh Kowdle, Vladimir Tankovich, Philip Davidson, and Shahram Izadi. 2017b. UltraStereo: Efficient Learning-based Matching for Active Stereo Systems. *CVPR* (2017).

Olivier Faugeras, Quang-Tuan Luong, and T. Papadopoulou. 2001. *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications.* MIT Press.

Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. 2009. Manhattan-World Stereo. *CVPR* (2009).

Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. 2016. Unsupervised cnn for single view depth estimation: Geometry to the rescue. *ECCV* (2016).

Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. 2017. Unsupervised monocular depth estimation with left-right consistency. *CVPR* (2017).

Google. 2018. ARCore - Google Developers Documentation. (2018). https://developers.google.com/ar

Rostam Affendi Hamzah and Haidi Ibrahim. 2016. Literature survey on stereo vision disparity map algorithms. *Journal of Sensors* (2016).

Richard Hartley and Andrew Zisserman. 2003. *Multiple view geometry in computer vision.* Cambridge university press.

Richard I Hartley and Peter Sturm. 1997. Triangulation. *Computer vision and image understanding* (1997).

Heiko Hirschmuller. 2008. Stereo processing by semiglobal matching and mutual information. *TPAMI* (2008).

Asmaa Hosni, Christoph Rhemann, Michael Bleyer, and Margrit Gelautz. 2011. Temporally consistent disparity and optical flow via efficient spatio-temporal filtering. *Pacific-Rim Symposium on Image and Video Technology* (2011).

Ramesh Jain, Sandra L Bartlett, and Nancy O'Brien. 1987. Motion stereo using ego-motion complex logarithmic mapping. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3 (1987), 356–369.

Olaf Kähler, Victor Adrian Prisacariu, Carl Yuheng Ren, Xin Sun, Philip Torr, and David Murray. 2015. Very high frame rate volumetric integration of depth images on mobile devices. *IEEE Transactions on Visualization and Computer Graphics* (2015).

Kevin Karsch, Ce Liu, and Sing Bing Kang. 2016. Depth Transfer: Depth Extraction from Videos Using Nonparametric Sampling. *Dense Image Correspondences for Computer Vision* (2016).

Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. 2017. End-to-end learning of geometry and context for deep stereo regression. *ICCV* (2017).

Sameh Khamis, Sean Fanello, Christoph Rhemann, Julien Valentin, Adarsh Kowdle, and Shahram Izadi. 2018. StereoNet: Guided Hierarchical Refinement for Edge-Aware Depth Prediction. In *ECCV*.

Hanme Kim, Stefan Leutenegger, and Andrew J Davison. 2016. Real-time 3D reconstruction and 6-DoF tracking with an event camera. *ECCV* (2016).

K. Klasing, D. Althoff, D. Wollherr, and M. Buss. 2009. Comparison of surface normal estimation methods for range sensing applications. *ICRA* (2009).

Kalin Kolev, Petri Tanskanen, Pablo Speciale, and Marc Pollefeys. 2014. Turning mobile phones into 3D scanners. *CVPR* (2014).

Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. 2016. Deeper Depth Prediction with Fully Convolutional Residual Networks. *CoRR* (2016). http://arxiv.org/abs/1606.00373

Ping Li, Dirk Farin, Rene Klein Gunnewiek, et al. 2006. On creating depth maps from monoscopic video using structure from motion. *IEEE Workshop on Content Generation and Coding for 3D-Television* (2006).

Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. 2016. Learning depth from single monocular images using deep convolutional neural fields. *IEEE transactions on pattern analysis and machine intelligence* 38, 10 (2016), 2024–2039.

Charles Loop and Zhengyou Zhang. 1999. Computing rectifying homographies for stereo vision. *CVPR* (1999).

Amrita Mazumdar, Armin Alaghi, Jonathan T. Barron, David Gallup, Luis Ceze, Mark Oskin, and Steven M. Seitz. 2017. A Hardware-friendly Bilateral Solver for Real-time Virtual Reality Video. *High Performance Graphics* (2017).

Mark Mine and Gary Bishop. 1993. Just-in-time pixels. *University of North Carolina at Chapel Hill Technical Report TR93-005* (1993).

Karsten Mühlmann, Dennis Maier, Jürgen Hesser, and Reinhard Männer. 2002. Calculating dense disparity maps from color stereo images, an efficient implementation. *IJCV* (2002).

Ramakant Nevatia. 1976. Depth measurement by motion stereo. *CGIP* (1976).

R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. 2011. DTAM: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*.

Manuel M. Oliveira, Brian Bowen, Richard Mckenna, and Yu sung Chang. 2001. Fast digital image inpainting. (2001).

Peter Ondrúška, Pushmeet Kohli, and Shahram Izadi. 2015. Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. *IEEE Transactions on Visualization and Computer Graphics* (2015).

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. 2016. Conditional image generation with pixelcnn decoders. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Curran Associates Inc., 4797–4805.

Liyuan Pan, Yuchao Dai, Miaomiao Liu, and Fatih Porikli. 2018. Depth Map Completion by Jointly Exploiting Blurry Color Images and Sparse Depth Maps. *WACV* (2018).

Jiahao Pang, Wenxiu Sun, JS Ren, Chengxi Yang, and Qiong Yan. 2017. Cascade residual learning: A two-stage convolutional neural network for stereo matching. (2017).

Jaesik Park, Hyeongwoo Kim, Yu-Wing Tai, Michael S Brown, and In So Kweon. 2014. High-quality depth map upsampling and completion for RGB-D cameras. *IEEE TIP* (2014).

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. 2016. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2536–2544.

Marc Pollefeys, Reinhard Koch, and Luc Van Gool. 1999. A simple and efficient rectification method for general motion. *ICCV* (1999).

Vivek Pradeep, Christoph Rhemann, Shahram Izadi, Christopher Zach, Michael Bleyer, and Steven Bathiche. 2013. MonoFusion: Real-time 3D Reconstruction of Small Scenes with a Single Web Camera. *ISMAR* (2013).

Malcolm Reynolds, Jozef Doboš, Leto Peel, Tim Weyrich, and Gabriel J Brostow. 2011. Capturing time-of-flight data with confidence. *CVPR*.

Christian Richardt, Douglas Orr, Ian Davies, Antonio Criminisi, and Neil A. Dodgson. 2010. Real-time Spatiotemporal Stereo Matching Using the Dual-Cross-Bilateral Grid. *ECCV* (2010).

Christian Richardt, Carsten Stoll, Neil A Dodgson, Hans-Peter Seidel, and Christian Theobalt. 2012. Coherent spatiotemporal filtering, upsampling and rendering of RGBZ videos. *Computer Graphics Forum* (2012).

Daniel Scharstein, Heiko Hirschmuller, York Kitajima, Greg Krathwohl, Nera Nesic, Xi Wang, and Porter Westling. 2014. High-Resolution Stereo Datasets with Subpixel-Accurate Ground Truth. *GCPR* (2014).

Daniel Scharstein and Richard Szeliski. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV* (2002).

Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. 2016. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision*. Springer, 501–518.

Thomas Schöps, Jakob Engel, and Daniel Cremers. 2014. Semi-dense visual odometry for AR on a smartphone. *ISMAR* (2014).

Thomas Schöps, Martin R Oswald, Pablo Speciale, Shuoran Yang, and Marc Pollefeys. 2017a. Real-Time View Correction for Mobile Devices. *IEEE Transactions on Visualization and Computer Graphics* (2017).

Thomas Schöps, Torsten Sattler, Christian Häne, and Marc Pollefeys. 2017b. Large-scale outdoor 3D reconstruction on a mobile device. *Computer Vision and Image Understanding* (2017).

Jianhong Shen and Tony F Chan. 2002. Mathematical models for local nontexture inpaintings. *SIAM J. Appl. Math.* (2002).

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. 2012. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*. Springer, 746–760.

Supasorn Suwajanakorn, Carlos Hernandez, and Steven M Seitz. 2015. Depth from focus with your mobile phone. *CVPR* (2015).

T. Taniai, Y. Matsushita, Y. Sato, and T. Naemura. 2018. Continuous 3D Label Stereo Matching using Local Expansion Moves. *PAMI* (2018).

Petri Tanskanen, Kalin Kolev, Lorenz Meier, Federico Camposeco, Olivier Saurer, and Marc Pollefeys. 2013. Live metric 3d reconstruction on mobile phones. *ICCV* (2013).

JMP Van Waveren. 2016. The asynchronous time warp for virtual reality on consumer hardware. *VRST* (2016).

Neal Wadhwa, Rahul Garg, David E. Jacobs, Bryan E. Feldman, Nori Kanazawa, Robert Carroll, Yair Movshovitz-Attias, Jonathan T. Barron, Yael Pritch, and Marc Levoy. 2018. Synthetic Depth-of-Field with a Single-Camera Mobile Phone. *SIGGRAPH* (2018).

Caihua Wang, H. Tanahashi, H. Hirayu, Y. Niwa, and K. Yamamoto. 2001. Comparison of local plane fitting methods for range data. *CVPR* (2001).

Chamara Saroj Weerasekera, Thanuja Dharmasiri, Ravi Garg, Tom Drummond, and Ian Reid. 2018. Just-in-Time Reconstruction: Inpainting Sparse Maps using Single View Depth Predictors as Priors. *arXiv preprint arXiv:1805.04239* (2018).

O. Woodford, P. Torr, I. Reid, and A. Fitzgibbon. 2009. Global Stereo Reconstruction under Second-Order Smoothness Priors. *TPAMI* (2009).

Chi Zhang, Zhiwei Li, Rui Cai, Hongyang Chao, and Yong Rui. 2014. As-Rigid-As-Possible Stereo under Second Order Smoothness Priors. *ECCV* (2014).

Dandan Zhang and Yuejia Luo. 2012. Single-trial ERPs elicited by visual stimuli at two contrast levels: Analysis of ongoing EEG and latency/amplitude jitters. *ISRA* (2012).

Yinda Zhang and Thomas Funkhouser. 2018. Deep Depth Completion of a Single RGB-D Image. *CVPR*.

Enliang Zheng, Enrique Dunn, Vladimir Jojic, and Jan-Michael Frahm. 2014. Patchmatch based joint view selection and depthmap estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1510–1517.

Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. 2017. Unsupervised learning of depth and ego-motion from video. *CVPR* (2017).

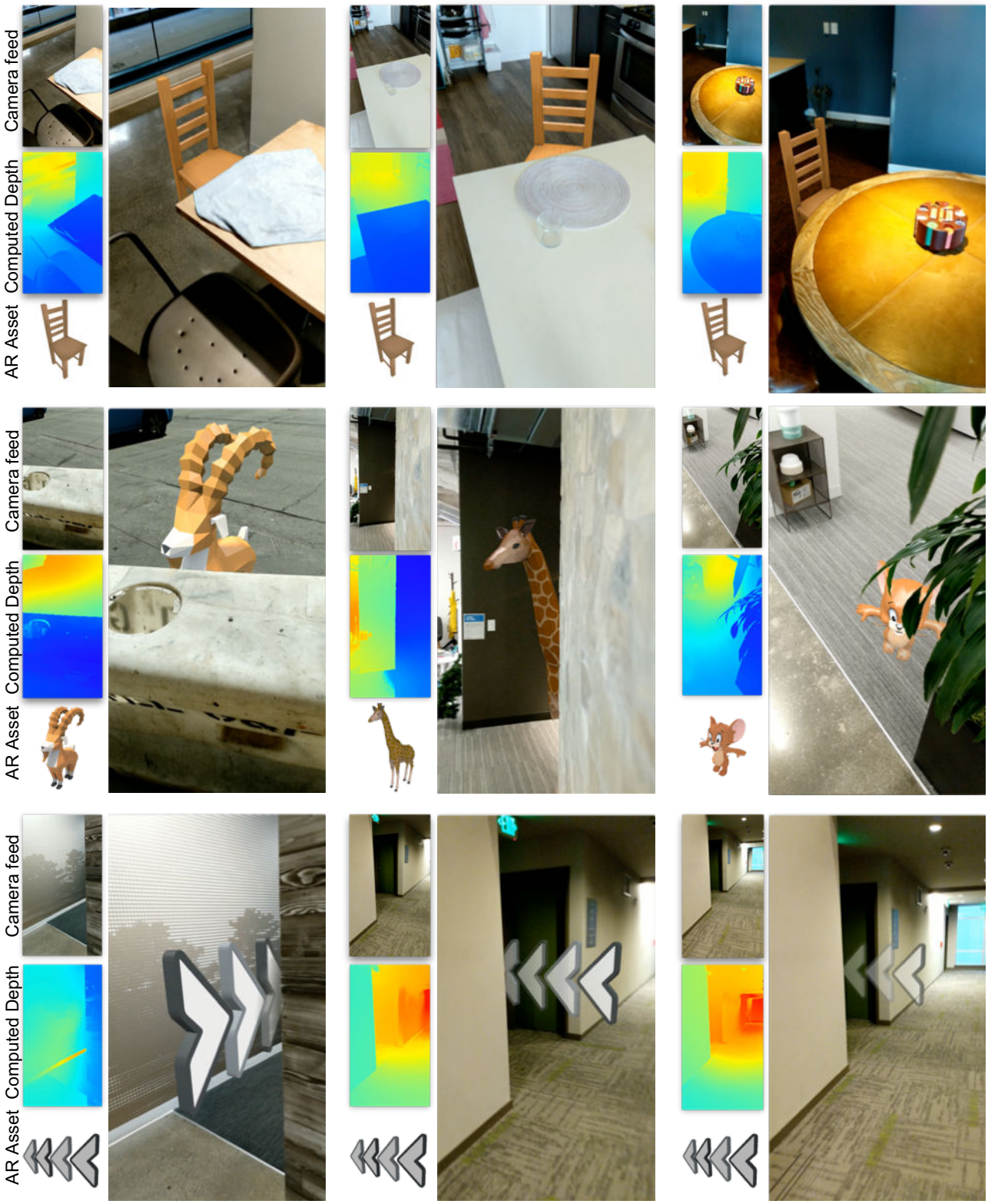


Fig. 20. Occlusion results. We show results for occlusion handling on a variety of challenging scenes for three different scenarios: online shopping (first row), fun photos/videos (second row) and navigation last row).