Passthrough+: Real-time Stereoscopic View Synthesis for Mobile Mixed Reality

GAURAV CHAURASIA, Facebook, Switzerland
ARTHUR NIEUWOUDT, Facebook, United States
ALEXANDRU-EUGEN ICHIM, Facebook, Switzerland
RICHARD SZELISKI, Facebook, United States
ALEXANDER SORKINE-HORNUNG, Facebook, Switzerland

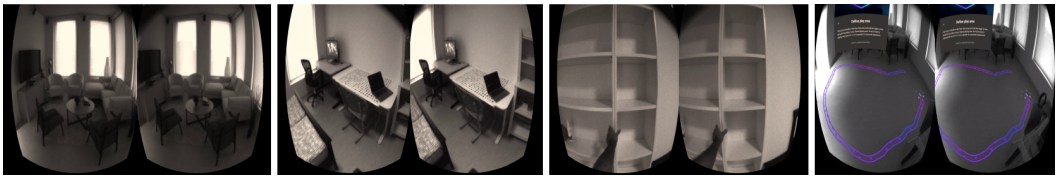

Fig. 1. We present stereoscopic view synthesis to display live first person viewpoints on VR devices. We use images from device cameras to compute a 3D reconstruction and warp the images using the 3D data to the viewpoint of user's eyes at display rate. From left to right: (a-b) our view synthesis is designed for indoor environments; we tested it on a wide range of layouts, lighting conditions, furniture etc. (c) This provides *haptic trust*; users can reach out and grab nearby objects guided by the stereoscopic view. (d) Our view synthesis is deployed on Oculus Quest VR devices as *Passthrough+* feature to allow users see their surroundings while they are marking a vacant *play area* in their room to enjoy VR content.

We present an end-to-end system for real-time environment capture, 3D reconstruction, and stereoscopic view synthesis on a mobile VR headset. Our solution allows the user to use the cameras on their VR headset as their eyes to see and interact with the real world while still wearing their headset, a feature often referred to as Passthrough. The central challenge when building such a system is the choice and implementation of algorithms under the strict compute, power, and performance constraints imposed by the target user experience and mobile platform. A key contribution of this paper is a complete description of a corresponding system that performs temporally stable passthrough rendering at 72 Hz with only 200 mW power consumption on a mobile Snapdragon 835 platform. Our algorithmic contributions for enabling this performance include the computation of a coarse 3D scene proxy on the embedded video encoding hardware, followed by a depth densification and filtering step, and finally stereoscopic texturing and spatio-temporal up-sampling. We provide a detailed discussion and evaluation of the challenges we encountered, as well as algorithm and performance trade-offs in terms of compute and resulting passthrough quality.

The described system is available to users as the *Passthrough+* feature on Oculus Quest. We believe that by publishing the underlying system and methods, we provide valuable insights to the community on how to design and implement real-time environment sensing and rendering on heavily resource constrained hardware.

Authors' addresses: Gaurav Chaurasia, gchauras@fb.com, Facebook, Giesshübelstrasse 30, 8045, Zürich, Switzerland; Arthur Nieuwoudt, arthurn@fb.com, Facebook, United States; Alexandru-Eugen Ichim, alex.ichim@fb.com, Facebook, Switzerland; Richard Szeliski, szeliski@fb.com, Facebook, United States; Alexander Sorkine-Hornung, alexsh@fb.com, Facebook, Switzerland.

2577-6193/2020/5-ART7
https://doi.org/10.1145/3384540

CCS Concepts: • **Computing methodologies → Mixed / augmented reality**; **Image-based rendering**.

Additional Key Words and Phrases: Mixed reality,augmented reality,image-based rendering,stereo reconstruction,video encoder,depth from motion vectors

ACM Reference Format:
Gaurav Chaurasia, Arthur Nieuwoudt, Alexandru-Eugen Ichim, Richard Szeliski, and Alexander Sorkine-Hornung. 2020. Passthrough+: Real-time Stereoscopic View Synthesis for Mobile Mixed Reality. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 1, Article 7 (May 2020), 17 pages. https://doi.org/10.1145/3384540

1 INTRODUCTION

Virtual Reality (VR) devices are becoming mainstream for gaming, media consumption, and productivity use-cases. This is evidenced by the growing ecosystem of content developers and providers, e.g., Oculus Store[1], SteamVR[2], and Netflix for VR[3]. The greatest strength of these devices is that they fully immerse the user into the content. This high level of immersion presents an important challenge: products that provide a fully immersive VR experience must also provide ways to help people remain aware of their surroundings and stay safe. In the absence of such functionality, the user has to pause the VR content and take the headset off for any interaction with the real world.

We present a complete solution for real-time reconstruction and stereoscopic view synthesis of the real-world environment, a feature commonly referred to as Passthrough, providing the user with a live feed of the surroundings. We leverage images from stereo pairs of cameras typically mounted on VR devices to enable positional tracking of the device pose using SLAM[4].

For a correct display of the user's surroundings, the cameras would have to be located at the user's eye positions. While this could be achieved with more complex optical systems, due to various manufacturing constraints, the cameras are usually located at the outer surface of VR headsets. Therefore, active reconstruction and warping of the camera images is required in order to provide a plausible and perceptually comfortable passthrough experience for the user. Specifically, our solution aims to provide the following:

(1) Plausible parallax without inducing motion sickness or visual discomfort.
(2) Low latency reconstruction and rendering without lag and stutter, while supporting dynamic scenes with significant motion.
(3) *Haptic trust*: users should experience natural proprioception, i.e., be able to reach out and grab an object or touch a surface.

Warping the images from the cameras to the user's eye positions using static geometry like a fixed plane or hemisphere without reconstructing 3D geometry is known to lead to instant motion sickness. Low latency is necessary for a smooth visual experience; stuttering rendering or judder can be disconcerting or disorienting to users, especially in fully immersive visual experiences where they have no other frame of reference. For *haptic trust*, it is important to refresh the scene geometry at a high enough rate to make individual geometry snapshots indiscernible to the eye, otherwise a nearby moving object will be perceived at an outdated position.

Our work addresses all of these challenges by combining concepts from 3D stereo reconstruction and image-based rendering (IBR). These fields are interestingly co-dependent. Stereo research has often used image interpolation to demonstrate the quality of depth maps, and IBR has focused on view synthesis from pre-captured scenes for which perfect depth maps cannot be computed

[1] https://www.oculus.com/experiences/quest/
[2] https://store.steampowered.com/steamvr
[3] https://play.google.com/store/apps/details?id=com.netflix.android_vr&hl=en_US
[4] All major VR headset brands currently provide such onboard stereo pairs, including Microsoft MR, Vive Pro, Google Daydream, and Oculus Quest/Rift-S.

(see Sec. 2 for details). Our work is among the first where real-time stereo is combined with IBR techniques for rendering live stereoscopic camera feed on a mobile, highly constrained platform.

In our application, we have to keep CPU utilization under 30 % of a single core and power consumption under 200 mW, so that our system can run smoothly without adversely affecting VR applications running in parallel, thermals, or battery health. Most sophisticated stereo techniques are not applicable in our case because they require much higher CPU utilization or a powerful GPU, to which we do not have access in our system. Therefore, we have developed an extremely low power stereo algorithm and depend upon IBR techniques to compute a warped camera feed that produces novel views at display rate, refreshes scene geometry at the rate images are captured without dropping input frames, and provides plausible parallax to users. Our main technical and system contributions include:

- an extremely low power stereo algorithm using consumer hardware (Sec. 3.1) and depth map computation (Sec. 3.2) to render a wide field of view in stereo (Sec. 3.4),
- novel algorithms for reinforcing temporal stability in rendered views (Sec. 3.3),
- a multi-threaded system design that can render novel views at display rate, irrespective of the underlying capture hardware (Sec. 4), and
- an analysis of end-to-end latency (Sec. 5) and reconstruction quality (Sec. 6) necessary for a comfortable stereoscopic experience.

2 PREVIOUS WORK

Our system uses a combination of real-time, low compute 3D reconstruction and novel view synthesis to warp the input images to the viewers' eye locations. In this section, we review previous work in these areas.

Real-time stereo reconstruction. Stereo matching is one of the most widely studied problems in computer vision [Scharstein and Szeliski 2002], but only small subset of algorithms are suitable for real-time implementation. An early example of such a system was the *Stereo Machine* of Kanade et al. [1996], which was implemented in custom hardware. Algorithms that were developed to have low computational complexity include semi-global matching [Hirschmüller 2008; Hirschmüller et al. 2012] and HashMatch [Fanello et al. 2017], which can compute real-time active illumination stereo on a GPU.

Valentin et al. [2018] use a combination of HashMatch and PatchMatch Stereo [Bleyer et al. 2011] to establish semi-dense correspondence between successive images in a smartphone augmented reality application. Like our system, they use consistency checks to eliminate unreliable matches, and then use a bilateral solver [Barron and Poole 2016; Mazumdar et al. 2017] to interpolate these correspondences to a full depth map, whereas we use a Laplace Solver [Di Martino and Facciolo 2018; Levin et al. 2004; Pérez et al. 2003]. Their paper also contains an extensive literature review.

Novel view synthesis. The study of *view interpolation*, which consists of warping rendered or captured images to a novel view, has been a central topic in computer graphics since its introduction by Chen and Williams [1993]. View interpolation was an early example of *image-based rendering* (IBR) [Chen 1995; McMillan and Bishop 1995], which more generally studies how to render novel views from potentially large collection of captured or rendered images [Shum et al. 2007][Szeliski 2010, Chapter 13]. This field is also often referred to as *novel view synthesis*.

A good early example of a complete system for real-time video view interpolation is the work of Zitnick et al. [2004]. In this system, multiple synchronized video cameras were used to record a dynamic scene from nearby viewpoints. The resulting videos were then processed *off-line* using a segmentation-based stereo algorithm [Zitnick and Kang 2007] to produce multi-layer depth maps

Fig. 2. Oculus Quest headset showing the position and orientation of cameras (blue) and the user's viewing direction (orange), and example images from the bottom two cameras. Due to the optical distortion and the positioning of the cameras vs. the user's eye positions, displaying the camera feed directly causes visual discomfort and motion sickness.

for each frame of each video stream. A real-time desktop rendering system could then interpolate in-between views to produce continuous viewpoint control as well as freeze-frame effects.

Since that time, this concept has been extended to more complex scenes and more dramatic view transitions. Stich et al. [2008] use local homographies with discontinuities to interpolate between cameras. Hornung and Kobbelt [2009] build 3D particle model for each view using multi-view stereo, then combine these at rendering time. Ballan et al. [2010] use billboards and view-dependent textures to interpolate between widely separated video streams, whereas Lipski et al. [2010] use dense correspondences between frames to interpolate them. Chaurasia et al. [2011] develop techniques for better handling of depth discontinuities ("silhouettes"), and Chaurasia et al. [2013] use super-pixel segmentation and warping plus hole filling to produce high-quality novel view synthesis in the presence of disocclusions.

More recently, Hedman et al. [2017] use the offline COLMAP system to do a sparse reconstruction, then use multi-view stereo with an extra near-envelope to compute a depth map on a desktop computer. Hedman and Kopf [2018] stitch depth maps from a dual-lens camera (iPhone) into a multi-layer panorama. Holynski and Kopf [2018] use a combination of DSO SLAM for sparse points, edge detection, and then edge-aware densification to compute high-quality depth maps on a desktop computer. Finally, the ESPReSSo system of Nover et al. [2018] computes real-time depth and supports viewpoint exploration on a desktop GPU using spacetime-stereo, i.e., 5 different IR illuminators, local descriptors, and PatchMatch. These systems produce high-quality results, but usually perform offline reconstruction and do real-time rendering on desktop computers.

The only system to date to demonstrate fully mobile depth computation (for augmented reality occlusion masking) is the one developed by Valentin et al. [2018] mentioned above. Unfortunately, their system is still too compute-heavy for our application, as we explain in the next section.

3 APPROACH

Our goal is to solve the following problem: *warp the images captured from a stereo camera pair on a VR headset in real-time to the user's eye positions* (Fig. 2). The synthesized views should enable a fluid, immersive viewing experience, with plausible parallax. All this has to happen on an extremely constrained platform: 30 % of a single mobile Qualcomm 835 CPU core at 200 mW power consumption.

Our overall algorithm from camera images to rendered views is shown in Fig. 3. Each of the algorithmic stages in the following subsections are designed to address the aforementioned criteria. The dominant parameter of our approach is depth map resolution, for which we chose 70×70 to fit the available GPU rasterization budget. Other parameters, e.g. weights in Sec. 3.2, 3.3, are tuned manually; we observed these performed well on meshes 4–6 times the resolution used in this paper.

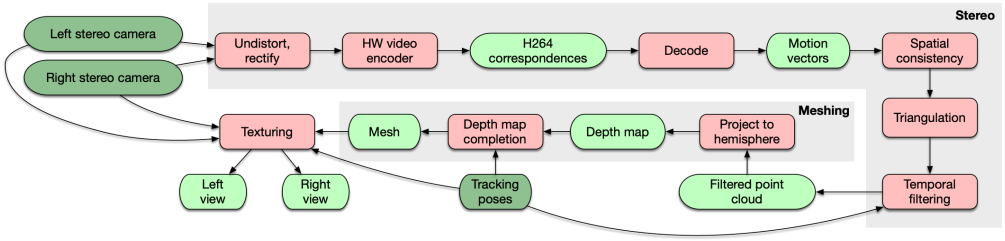

Fig. 3. Algorithmic overview. Starting from input images, we first rectify the images and feed them to the hardware video encoder, from which we extract and filter *motion vectors* (Sec. 3.1). The resulting correspondences are converted into depth values, to which we apply temporal refinement (Sec. 3.3). We project and densify the points on to a wide field of view hemispherical grid centered at user's position (Sec. 3.2). Finally we create a mesh with associated texture coordinates corresponding to left and right input images, which is then rendered to the left and right eye views (Sec. 3.4).

3.1 Low power stereo reconstruction

Our view synthesis starts with a sparse, low-power 3D reconstruction of the scene. Given a stereo camera pair on a VR device, we compute stereo correspondences, which we then triangulate into 3D points after a series of consistency checks. Traditional techniques such as dense semi-global stereo matching [Hirschmüller 2008] require a high CPU load even with vectorized implementations, and are not feasible for our use case and platform constraints. In order to meet the power and latency requirements, we exploit the video encoding hardware available on mobile SoCs to compute correspondences or *motion vectors* (MVs). Mobile chipsets like our Qualcomm 835 have custom silicon for video and audio encoding, which operate on a much lower power budget than CPUs: 80 mW compared to 1000 mW for a CPU.

Encoder stereo. Video encoders are designed to compute correspondences from frame t to $t+1$ of an input video sequence, and return these MVs in an encoded video stream at a very low power budget. We re-purpose the encoder to compute motion vectors between the left and right images of the stereo pair, instead of consecutive video frames. See Fig. 3 for an overview of our full 3D reconstruction approach.

Sending the original input stereo pair to the video encoder does not provide useful results, since encoders are usually biased towards small MVs via block-based motion estimation [Jakubowski and Pastuszak 2013]. This reduces the ability to detect large displacements due to close-by objects. Moreover, they operate on macro-blocks of size 8×8 [Ostermann et al. 2004], which we found to be too coarse for our application.

To overcome these limitations, we create the input to the video encoder as a mixture of transformations of the left and right rectified images. In this mosaic, we arrange multiple copies of the input images to force correspondence estimation at an offset equal to half the macro-block size (i.e., 4 pixels), so as to obtain sub-macro-block matching (Fig. 4). We also pre-shift the left subframe to the left, which increases the probability of detecting large disparities. For example, a pre-shift of 4 pixels places a true 12 pixel original disparity at 8 pixels where it is more likely to be found by the video encoder. We use multiple transformations with pre-shifts of 8 and 32 pixels to cover a wide range of possible disparities. Secondly, we shift the macro-block grid by half the macro-block size. Thus, if the encoder operates on a grid spaced by 8 pixels, we create 4 subframes with a shift of 4 pixels (Fig. 4). This allows us to compute MVs at a spacing of 4 pixels.

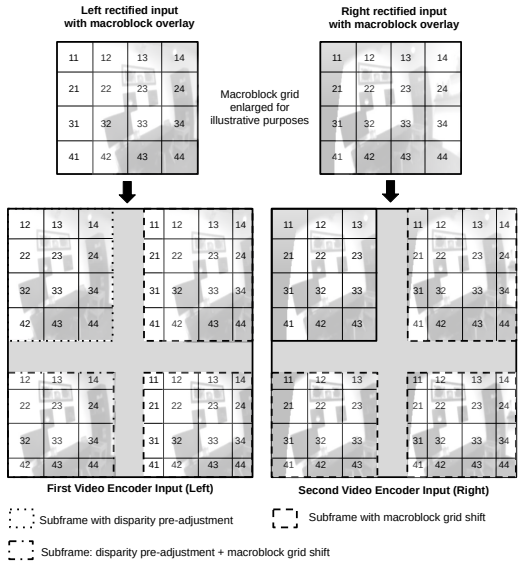

Fig. 4. Video encoder input mosaic showing disparity pre-adjustment and macro-block grid shift. Disparity pre-adjustment translates the left frame so that the video encoder can find large disparities at smaller translation. The macro-block grid shift forces 4 motion vectors to be computed per macro-block instead of 1.

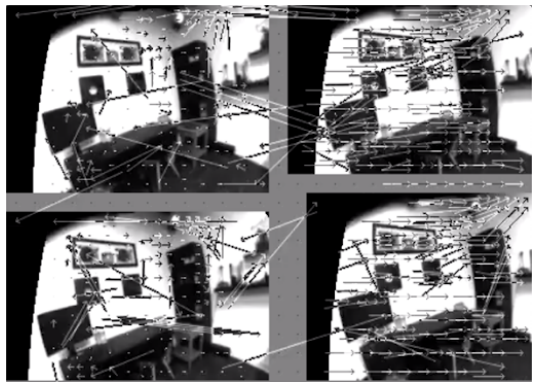

Fig. 5. Motion vectors computed by the encoder. Many of the correspondences are noisy and have to be discarded via spatial consistency checks and temporal reinforcement.

In addition, encoders require a number of parameters to be tuned, such as bit rate or I-block period, which are discussed in Sec. A. Ostermann et al. [2004] provide additional details on H.264 encoding.

Spatial consistency checks. Motion vectors from the encoder exhibit significant noise (Fig.5) because they do not undergo regularization [Hirschmüller 2008]. For each point in the left image for which we have a motion vector, we apply the spatial consistency checks listed in Table 4 in the appendix. The most important is the left-right consistency check: we compute motion vector for left to right image and also from right to left image with the same mosaics that we computed earlier (Fig. 4). The motion vectors that pass all these consistency checks represent the final set

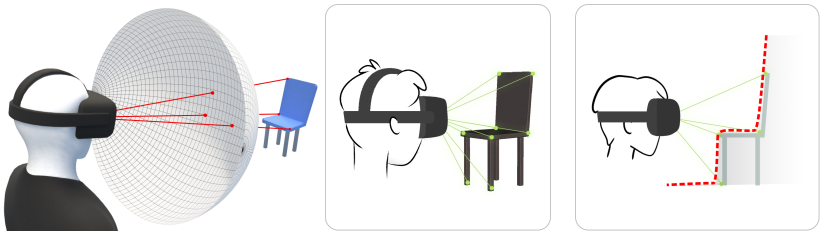

Fig. 6. 180° field of view mesh centered at the user (left) with projection of 3D points from motion vector stereo. We use 3D points (middle) as constraints in a Laplace equation (Sec. 3.2) to deform the unit distance hemispherical mesh to the shape of the object (right, red).

of valid correspondences, which we turn in into 3D points via triangulation. As described later in Sec. 3.3, we apply temporal filtering on this final set of points to reinforce stability.

Overall, the above approach produces 300–1200 3D points. The total run time is around 7 ms, a large fraction of which is spent on the video encoder (Table 2). In comparison, a simple patch-based stereo matching approach without any regularization required 10 ms on the CPU to yield a comparable number of points after our best efforts to vectorize the computation.

3.2 From motion vector stereo to dense depth

The unstructured, relatively sparse set of 3D points from the stereo algorithm (Sec. 3.1) is insufficient for high quality passthrough due to their non uniform distribution, noise, and outliers. In order to perform high quality view synthesis for passthrough, we need to convert these 3D points into a scene proxy that provides dense and stable per-pixel depth and covers a wide field-of-view to account for fast user motion and head rotation. We solve this in a densification step by filtering and propagating sparse depth.

Basic approach. We create a virtual hemisphere at unit distance around the user (Fig. 6, left). The hemisphere is parameterized using Euler angles, representing it as a $n \times n$ (70×70 in our experiments) grid of cells. Projecting 3D points onto the hemisphere results in depth values for corresponding cells, effectively turning it into a wide field-of-view *depth map*.

In order to fill in empty regions on the hemispherical depth map, values from cells that have associated depth have to be propagated across the grid. This is conceptually similar to heat diffusion inspired solutions for propagating color strokes in order to colorize a grayscale image [Levin et al. 2004]. We therefore use the Laplacian operator to propagate depth information across the grid:

$$\arg\min_{x} \|\mathbf{L} \cdot \mathbf{x}\|^2 + \lambda \sum_i w_i \, \|x_i - \bar{x}_i\|^2, \tag{1}$$

where $\mathbf{L}$ is the Laplacian operator, $\mathbf{x}$ is the grid of unknown inverse depth values for each cell of the hemisphere arranged a column vector. w_i is the sum of weights of all 3D from motion vector stereo that project to the i-th cell in the hemisphere, and $\bar{x}_i$ is the weighted mean of known inverse depth values of all 3D points that project to the i-th cell in the hemisphere. Each 3D point computed from the current stereo pair is added with a constant weight, set to 5 in our current implementation, such that three points projecting into the i-th cell results in $w_i = 15.0$. In Sec. 3.3 we describe how this weighting scheme can be used to achieve improved temporal stability by adding in 3D points from previous frames with lower weights. We initialize the border of the depth map to a plausible fixed depth value of 2.0 m as Dirichlet border constraints. All the operations are performed on inverse depth [Goesele et al. 2010].

Table 1. Median number of iterations and wall clock time for Conjugate-Gradient (CG) and Jacobi over-relaxation (JAC).

Relative tolerance	CG iterations	JAC iterations	CG time	JAC time
1×10^{-6}	29	111	5.07 ms	4.15 ms
1×10^{-5}	16	37	2.8 ms	1.3 ms
1×10^{-4}	6	7	1.05 ms	0.25 ms

Performance optimization. A straightforward implementation of this approach unfortunately has prohibitive complexity, given our extremely tight compute and power budget. We therefore evaluated various approaches and solve the problem as follows. Our equation is a simpler case of the Poisson equation; hence research into high performance Poisson solvers for regular lattices is applicable. We experimented with a direct Cholesky solver, an iterative Conjugate-Gradient solver, and a vectorized implementation of iterative Jacobi over-relaxation [Di Martino and Facciolo 2018]. The direct solver was unsurprisingly the slowest. We aided the other two iterative solvers by initializing them with the depth map from the previous frame. Conjugate-Gradient has a better convergence rate than the Jacobi solver (Table 1); Hierarchical Basis Preconditioners [Szeliski 2006] could further reduce the number of iterations. However, we found that vectorized Jacobi over-relaxation makes better usage of memory locality; it was around 5× faster for time per iteration (0.037 ms compared to 0.175 ms for Conjugate-Gradient). In terms of overall time for 1×10^{-5} relative tolerance, Jacobi over-relaxation was 2× faster.

Note that we do not incorporate image edges as a constraint in Eq. (1). Levin et al. used pixel differences as weights; later depth propagation approaches have incorporated either explicit silhouette awareness [Chaurasia et al. 2013] or an edge-aware bilateral solver [Barron and Poole 2016; Mazumdar et al. 2017; Valentin et al. 2018]. The reason is that, due to limited GPU availability on a VR device, we restrict the resolution of the depth map 70×70. At such low resolutions, the size of a grid cell in the hemisphere usually spans across multiple object boundaries, and adding edge weights therefore did not add any value. If more CPU/GPU resources were available, incorporating edge weights could produce a higher better fidelity of the depth map at object boundaries. However, since the 3D points are warped over a short baseline from camera pose to eye pose, and we are exploiting stereoscopic texturing during rendering, a plausible visual reproduction of (dis-)occlusion artifacts can be achieved without edge constraints.

3.3 Temporal filtering

The depth map in the previous section is computed by the discretization and regularization of sparse 3D points. If done independently per camera frame, temporal changes due to camera noise, lighting conditions, or dynamic scene content cause considerable fluctuations in the point set. These fluctuations make their way into the depth map and cause noticeable warping, bending, and wiggling in the Passthrough rendering. We alleviate this by temporally filtering both the sparse 3D points and the completed depth map. Joint spatio-temporal optimization [Davis et al. 2003; Richardt et al. 2010] is not an option because it leads to prohibitively high CPU usage and power budget. Our approach works as follows.

Temporal reinforcement of stereo points. Instead of directly using points from stereo to compute the depth map, we compare them to a set of stereo points from past frames, which serves as a *prior* set. Over time, this results in a set of stereo points that have been observed repeatedly.

We identify three sets of 3D points in world coordinates:

- $\mathcal{S}_{\text{obv}} \equiv \{(p_i, d_i)\}$ *observation*: stereo points p_i with disparity values from current images (Sec. 3.1),
- $\mathcal{S}_{\text{prior}} \equiv \{(p_i, w_i)\}$ *prior* : points from built over previous frame(s), each with a weight w_i, and
- $\mathcal{S}_{\text{res}} \equiv \{p_i\}$ *result*: output points.

Here, disparity value for a 3D point is the difference between its projection in the right and left rectified images at a particular instant. At any frame, we have the points $\mathcal{S}_{\text{obv}}$ from stereo on current images, and a prior $\mathcal{S}_{\text{prior}}$ built over previous frames. Two points in $\mathcal{S}_{\text{prior}}$ and $\mathcal{S}_{\text{obv}}$ are considered *matching* if their projection in the current right image and the corresponding disparity values are within 2 pixels. Using this definition, we compute all the points in $\mathcal{S}_{\text{prior}}$ which have a match in $\mathcal{S}_{\text{obv}}$. We increment the weight w_j of the subset of $\mathcal{S}_{\text{prior}}$ for which a match could be found in $\mathcal{S}_{\text{obv}}$. Analogously, we decrease the weights of those for which no match was found. The points in $\mathcal{S}_{\text{obv}}$ which did not coincide with any point $\mathcal{S}_{\text{prior}}$ are added to $\mathcal{S}_{\text{prior}}$ with weight 0, where "coincide" means that points from $\mathcal{S}_{\text{prior}}$ and $\mathcal{S}_{\text{res}}$ project within 1 pixel of each other in the right rectified image. All points in $\mathcal{S}_{\text{prior}}$ whose weights are greater than 0 are additionally advanced to the result set $\mathcal{S}_{\text{res}}$ for current instant, and those whose weights dropped below 0 are removed from $\mathcal{S}_{\text{prior}}$. Our algorithm can thus be summarized as:

$$\begin{aligned} \mathcal{S}_{\text{res}} &\leftarrow \mathcal{S}_{\text{prior}} \cap \mathcal{S}_{\text{obv}}, \\ \mathcal{S}_{\text{prior}} &\leftarrow \mathcal{S}_{\text{prior}} \cup \mathcal{S}_{\text{obv}}, \\ \mathcal{S}_{\text{prior}} &\leftarrow \mathcal{S}_{\text{prior}} \setminus \mathcal{S}_{\text{zero}}. \end{aligned}$$

where $\mathcal{S}_{\text{zero}} = \{(p_j, w_j) | w_j < 0\}$.

We start with empty $\mathcal{S}_{\text{prior}}$, $\mathcal{S}_{\text{obv}}$ and $\mathcal{S}_{\text{res}}$. At the first frame, since the prior $\mathcal{S}_{\text{prior}}$ is empty, none of the stereo points $\mathcal{S}_{\text{obv}}$ can be advanced to $\mathcal{S}_{\text{res}}$, but they get added to $\mathcal{S}_{\text{prior}}$. At the second frame, the newly computed $\mathcal{S}_{\text{obv}}$ can be compared against $\mathcal{S}_{\text{prior}}$; the matching subset of $\mathcal{S}_{\text{prior}}$ is advanced to $\mathcal{S}_{\text{res}}$, and $\mathcal{S}_{\text{prior}}$ itself is augmented with $\mathcal{S}_{\text{obv}}$. This achieves two important results:

- **Outlier suppression.** 3D points need to be observed in at least two frames to advance to $\mathcal{S}_{\text{res}}$ and influence the depth map. This removes a vast majority of spurious points.
- **Temporal history of prominent points.** A point gets advanced to $\mathcal{S}_{\text{res}}$, if its weight remains greater than 0. Moreover, the weight of a point in $\mathcal{S}_{\text{prior}}$ gets repeatedly incremented every time it is observed in $\mathcal{S}_{\text{obv}}$. This way point can be passed across longer temporal windows. If they are not observed in one frame, they can still be used from the prior set. If the point is out of field of view, its weight gets decremented automatically since it is no longer in $\mathcal{S}_{\text{obv}}$. After a while, it is completely dropped when its weight drops below 0.

In practice, we restrict the maximum weight of points in $\mathcal{S}_{\text{prior}}$ to 6 for nearby objects closer than 1 m. This ensures that for fast movements of nearby scene elements such as hands, stale geometry that is older than 6 frames (equivalent to 0.2 s for camera feed) does not negatively affect the rendering quality. For geometry further away, we keep the threshold at 16 (0.5 s of camera feed).

Temporal smoothing of the depth map. The final depth map $\mathbf{x}_t$ at time t is computed by solving the Laplace equation with stereo points as constraints (Sec. 3.2). Even though the previously described temporal reinforcement suppresses most of the temporal outliers, some temporal instability is still caused by the discretization of the 3D points to a low resolution depth map in a time varying manner. We alleviate this by projecting the depth map $\mathbf{x}_{t-1}$ from the previous frame to the current headset pose T and using the resulting 3D points as additional constraints in the solver. Hence we

replace

$$\mathbf{x}_t \leftarrow \texttt{LaplaceSolver}\left[\mathcal{S}_{\text{res}}(t)\right],$$

from (Sec. 3.2) by

$$\mathbf{x}_t \leftarrow \texttt{LaplaceSolver}\left[\mathcal{S}_{\text{res}}(t) \ \cup \ T \cdot \mathbf{x}_{t-1}\right].$$

This is inspired by Infinite Impulse Response (IIR) filters, wherein a signal at time t is filtered with input t and its own value at $\{t-1, \cdots\}$ to approximate a decaying filter over an infinite window.

In all our experiments, this was significantly more effective than caching stereo points from last $k < 10$ frames and using them as constraints, because it allowed much longer temporal aggregation with a fixed number of values to store and track. Caching stereo results from multiple frames quickly led to a prohibitive increase in memory and CPU cycles to store, project and incorporate the points as constraints.

Recall that in Eq. 1, we had used Laplacian constraints with weight 1.0 and 3D constraints with weight 5.0, accumulated into the weight matrix $\mathbf{W}$. Since we now have 3D constraints from stereo points $\mathcal{S}_{\text{res}}(t)$ and from previous depth map $\mathbf{x}_{t-1}$, we specify different weights for these. We use a standard weight of $\frac{5.0}{30.0}$ for each point in $\mathcal{S}_{\text{res}}(t)$. For points from $\mathbf{x}_{t-1}$ projected into current frame, we use a weight of 5.0 if that it was within one grid cell from the projection of stereo point $\mathcal{S}_{\text{res}}(t-1)$ in the previous frame's depth map, else 0.0. We prioritize previous frame's output 30 times more than current stereo points – this is important for strong temporal stabilization. As this prioritization is decreased, temporal stabilization weakens.

3.4 Stereoscopic rendering

The final depth map values are sent to the GPU and rendered as a triangle strip. All transformations are performed in a vertex shader to minimize CPU-GPU bandwidth. We project the mesh vertices into the left and right input cameras to obtain individual left and right texture coordinates, which are used to texture the mesh using the left/right camera for the left/right eye, respectively.

This view-dependent stereoscopic texturing has the advantage that each of left and right input views are warped over a shorter baseline as compared to warping a single image to both eyes, as well as preserving view-dependent illumination and parallax effects, resulting in an overall more plausible rendering. Another advantage is that this fills up the entire 180° field of view around the user; which would not be possible using the same image to texture the mesh for both eyes.

The caveat of this method is that occasionally, each eye views a different texture on the same fragment of 3D geometry, which the visual system cannot merge. We observed that this is noticeable only for objects around 30 cm; 3D reconstruction is also often inaccurate at such close ranges (Fig. 9b).

4 SYSTEM ARCHITECTURE

Our system consists of a VR headset that has a stereo pair of cameras and is capable of computing device pose in world coordinates at high precision, typically using a SLAM system.

We implemented our system on two threads: a render thread and a compute thread, see Fig. 7. The render thread is triggered every time the display is refreshed. Inside the render loop, we get the real-time device pose from the SLAM system. We poll the cameras for new images. If new images are available, we first upload them to the GPU for rendering. If the compute thread is not currently busy, we also copy the images locally for processing and signal the compute thread to wake up. The compute thread generates a new depth map from images as described in Sec. 3. It uses double buffered output: one of these buffers is for the compute thread to overwrite while the other is read on the render thread. These buffers are swapped and marked fresh every time the compute thread has a new output.

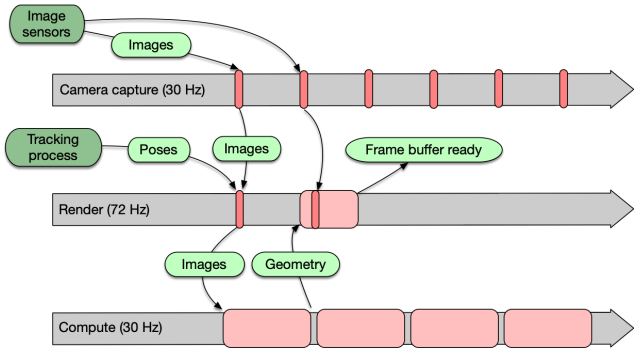

Fig. 7. Multi-threaded system using two main threads, with an additional camera capture thread operating synchronously at 30 Hz. The render thread triggers synchronously at 72 Hz. It copies the images when available, and triggers and polls the compute thread for geometry, which it uses to compute texture coordinates. Finally it uploads textures and attributes to the GPU and renders the mesh.

Back on the render thread, we query the compute thread for a new depth map. If available, it is uploaded to the GPU; otherwise we retain the old depth values already in GPU memory. Every time we either have a new image or new depth map, we recompute texture coordinates for the updated mesh using the device pose at the time the images were captured. We observed that an error of even 4 ms in capture timestamps was enough to cause visibly floating and lagging rendering. The SLAM system must be capable of delivering device pose at such a high frame rate and also with high accuracy.

An important consideration is to never block the render thread for a long (>4 ms) or non-deterministic period. Therefore, we only perform the non-blocking fast operations on this thread (Table 3) to fit well within the 14 ms between two successive display refresh events. We minimize CPU-GPU bandwidth by only sending depth values to the GPU instead of 3D vertices; these depth values are multiplied to unit bearing vectors (uploaded once to GPU) in the OpenGL vertex shader to get 3D vertex coordinates.

5 PERFORMANCE

We deployed our system on the Oculus Quest VR headset, which has a mobile Qualcomm 835 processor. Quest devices have a downward facing grayscale stereo camera pair (Fig. 2) that operates at 30 Hz. The display refresh rate is 72 Hz. It also has a SLAM system[5] for real-time head pose estimation. We aggregated performance numbers over 10 000 h of usage in uncontrolled conditions.

The Qualcomm 835 CPU runs at 2.45 GHz. The execution times for all the stages are given in Table 2 and 3. Our multi-threaded solution (Sec. 4) maintains a display rate of 72 Hz irrespective of the time spent in the compute thread.

For our end-to-end multi-threaded system, the delay between image capture and the time when the corresponding camera image is rendered is 49 ms. The delay between the image capture and the rendering corresponding 3D geometry is 62 ms. These take into account the delay between camera and display events, as well as thread scheduling, in addition to the computation time. We call these *photon-to-texture* and *photon-to-geometry* latency, respectively. These determine the responsiveness of our system. We measure these using the mid point of the cameras' exposure window as the start time, and the time when texture/geometry are rendered in an OpenGL context

[5] https://ai.facebook.com/blog/powered-by-ai-oculus-insight/

Table 2. CPU execution time for various stages in the compute thread. †CPU is idle during motion vector computation on the video encoder.

Stereo computation		**Depth map completion**	
Rectification	0.6 ms	System matrix setup	0.8 ms
Motion vectors†	2.8 ms	Solver	0.7 ms
Decoding	1.7 ms		
Spatial checks	0.5 ms		
Temporal filtering	1.0 ms		
Total			< 9 ms

Table 3. CPU execution time for the render thread.

Render thread	
Copy images	0.3 ms
Compute texture coords	2.2 ms
Send textures to GPU	0.5 ms
Send depth values to GPU	0.1 ms
Send texture coords to GPU	0.2 ms
Total	< 4 ms

as end time. Getting photon-to-texture latency as close as possible to 33 ms (corresponding to 30 Hz) is important; a latency higher than around 100 ms can cause noticeable judder. In our experiments, photon-to-geometry latency requirements seemed less critical; some of our early prototypes had this latency above 100 ms without too much additional discomfort.

We also implemented and tested our approach on the PC-based Oculus Rift-S[6], which has a different camera arrangement, with similar results.

6 EVALUATION

We evaluate the quality of the described system from two main perspectives: a quantitative comparison of the depth estimation to high quality reference depth maps from laser scans, and an ablation study on temporal coherence, which is one of the key factors in perceived rendering quality. Depth maps from laser scans may have small errors, hence we refer to them as *pseudo* ground truth.

6.1 Depth Estimation

We evaluated our depth estimation on 450 datasets, amounting to 9 hours of recordings, on Quest headsets in a variety of indoor scenes. We captured egocentric datasets while walking around indoor scenes, performing tasks like game play, watching videos etc. We evaluate two key metrics that influence the final rendering:

- **Absolute depth error.** This is the difference in pseudo ground truth depth and calculated depth of a stereo point. We aggregate this per-point by depth ranges: we group depth measurements are grouped into a histogram with an interval width of 0.25 m and plot the depth error of each histogram bucket. This gives depth error as a function of absolute depth value.

[6] https://www.oculus.com/rift-s/

- **Image coverage.** This is the percentage of the input image where 3D reconstruction was successful. We divide the image into blocks of 20×20 and count the number of blocks that have at least one stereo point. Sparse coverage implies parts of the scene were not reconstructed and parallax will likely be poor for rendering such regions. Naturally, higher coverage can be expected to include 3D points with high depth errors as well. Hence, we plot this metric as a function of maximum allowed depth error.

Before we describe the results, we outline the process for acquiring pseudo ground truth depth maps.

Fig. 8. Datasets. Input view (left) and corresponding pseudo ground truth depth map from a laser scan (right).

Pseudo ground truth depth maps. We first select a static scene configuration. We capture multiple high-precision laser scans and photographs of the scenes. The scans are stitched together to form a detailed textured mesh. Next, we use VR devices to record egocentric videos in the pre-scanned scene. We also capture precise per-frame pose, tracking the headset with an OptiTrack system[7]. We align the laser scan and the device poses captured by OptiTrack using physical beacons that can be identified uniquely in both laser scans and cameras. After registration, we have the laser scans, camera trajectories and the corresponding videos in the same coordinate frame. Finally, we generate pseudo ground truth depth maps for each video frame by rendering the textured mesh from the device's camera viewpoint.

Absolute depth error. Fig. 9b shows that error increases steadily with absolute depth; it is between 5 cm–10 cm for most of the scene content. An important contribution of our work is highlighting that this level of accuracy can be expected to give comfortable visual experience. Note the higher errors at depths closer than 1 m. This is because our stereo algorithm is biased towards smaller disparities (Sec. 3.1). Improving reconstruction of close-by objects is one of our focus points for future work.

Image coverage. Fig. 9a shows that irrespective of the depth accuracy threshold, the coverage is constant. That means that our stereo algorithm produces points whose accuracy is uniformly distributed, and that the approach is not biased towards any particular accuracy limit. This is important for productization: we can set a maximum limit, e.g. 2000, 10 000 etc., points per-frame and expect the same accuracy distribution. This allows us to design downstream heuristics like depth map completion without overfitting to heuristics in the stereo algorithm. Fig. 9d shows the average number of points generated per frame for each depth range, with our current settings.

[7] https://www.optitrack.com/

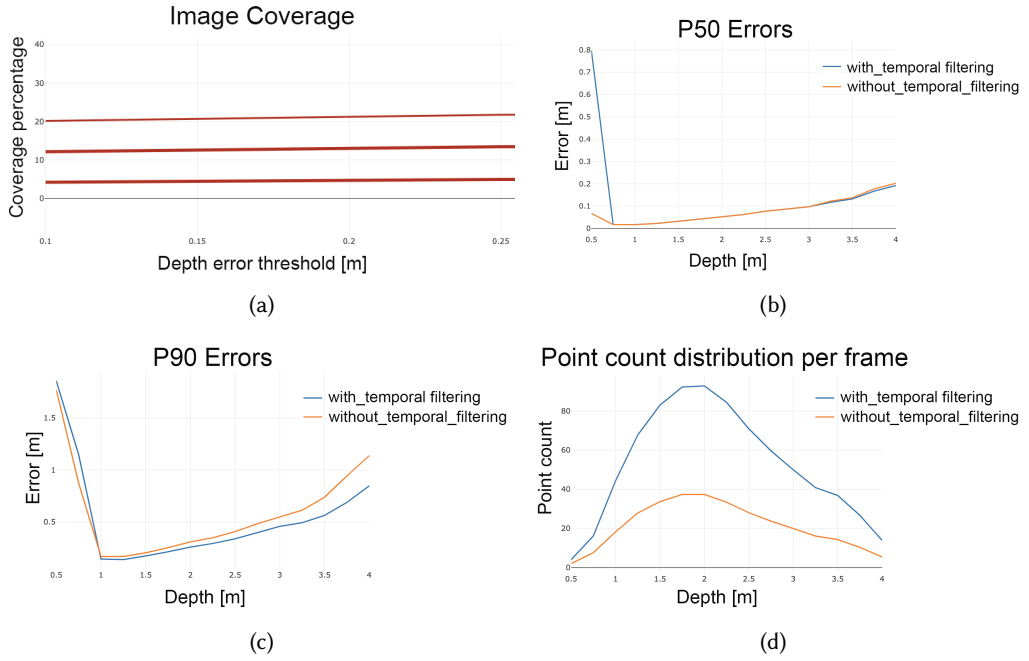

Fig. 9. Evaluation for depth estimation. (a) Image coverage (y) as a function of maximum allowed depth error (x) in meters. (b) Median depth error (y) as a function of depth (x) with and without temporal reinforcement. (c) 90^{th} percentile depth error (y) as a function of depth (x); temporal reinforcement causes this to drop indicating significantly fewer outliers. (d) Point count as a function of depth (x); temporal reinforcement reuses points from multiple previous frames without incurring outliers.

Effect of temporal reinforcement of stereo points. This step (Sec. 3.3) does not change the median depth accuracy (Fig. 9b), but 90^{th} percentile errors drop significantly (Fig. 9c), which proves the efficacy of this step in removing depth outliers. This is important since spurious 3D points appearing and disappearing across time leads to deformed structures in the rendered result. Moreover, number of healthy 3D points in each frame is also higher (Fig. 9d), since this heuristic allows us to reuse points from previous frames.

Effect of temporal constraints on depth map completion. We demonstrate this on two live recordings. The recording without temporal constraints shows high frequency wiggling. This is because the 3D points are discretized on the lower resolution depth map grid (Sec. 3.2). This is alleviated when we add temporal constraints in the Laplace solver. Please refer to accompanying video for this comparison.

7 LIMITATIONS

The most important limitation of our approach is that the mesh can be inaccurate for nearby objects and does not align with object boundaries, as the depth estimation around 30 cm becomes inaccurate (Fig. 9b), and correspondences from the video encoder are on a fixed grid of macro-blocks and get discarded when ending up on textureless image regions (Sec. 3.1). This can cause object boundaries to occasionally wobble and limit stereoscopic viewing comfort for very close-by objects (also explained in Sec. 3.4). This can also lead to occlusion artifacts when used in an augmented

reality context to place virtual objects. We have explored efficient CPU based stereo to alleviate this. Moreover, hardware vendors are starting to provide hardware accelerated correspondence estimation[8]. These would still not be as high quality as SGM [Hirschmüller 2008], but at least provide correspondences in each scene object, which, when combined with an edge aware solver (instead of Sec. 3.2), should be able to capture object boundaries in the depth map.

8 CONCLUSION

We have presented a real-time system to reconstruct coarse 3D geometry and display realistic renderings of the real world captured from cameras that do not coincide with user's eyes. To this end, we adapted a number of well known techniques in computer vision and graphics to achieve a delicate balance between resource consumption, 3D accuracy and visual fidelity so as to maximize user experience. Our work demonstrates how to build basic environment sensing ability into the current generation of VR headsets.

Future work. VR headsets offer a great opportunity to push the boundary of real-time scene understanding. In this work, we focused on 3D reconstruction and view synthesis to allow a VR user to become aware of their surroundings. We are currently investigating the effect of 3D reconstruction accuracy on visual perception and how to achieve higher haptic trust and visual fidelity. In the future, we will apply environment sensing technologies such as object detection and recognition to not only enhance safety in VR, but to also explore user experiences that eventually blur the boundary between the real and virtual world.

ACKNOWLEDGMENTS

Developing this system has been a significant effort across many teams and disciplines. The authors would like to thank all team members and the Facebook XR Tech management for their support. Special thanks to Alessia Marra for providing explanatory diagrams.

REFERENCES

Luca Ballan, Gabriel J. Brostow, Jens Puwein, and Marc Pollefeys. 2010. Unstructured Video-Based Rendering: Interactive Exploration of Casually Captured Videos. *ACM Trans. Graph. (Proc. SIGGRAPH)* 29, Article 87 (July 2010), 11 pages. Issue 4. https://doi.org/10.1145/1778765.1778824

Jonathan T. Barron and Ben Poole. 2016. The Fast Bilateral Solver. In *The European Conference on Computer Vision (ECCV)*.

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

Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. 2013. Depth Synthesis and Local Warps for Plausible Image-based Navigation. *ACM Trans. Graph.* 32, 3, Article 30 (June 2013), 12 pages. https://doi.org/10.1145/2487228.2487238

Gaurav Chaurasia, Olga Sorkine, and George Drettakis. 2011. Silhouette-Aware Warping for Image-Based Rendering. *Comput. Graph. Forum (Proc. EGSR)* 30, 4 (2011), 1223–1232. https://doi.org/10.1111/j.1467-8659.2011.01981.x

S. Chen and L. Williams. 1993. View Interpolation for Image Synthesis. In *ACM SIGGRAPH 1993 Conference Proceedings*. 279–288.

S. E. Chen. 1995. QuickTime VR – An Image-Based Approach to Virtual Environment Navigation. In *ACM SIGGRAPH 1995 Conference Proceedings*. 29–38.

J. Davis, R. Ramamoorthi, and S. Rusinkiewicz. 2003. Spacetime stereo: a unifying framework for depth from triangulation. In *CVPR*, Vol. 2. II–359. https://doi.org/10.1109/CVPR.2003.1211491

Matías Di Martino and Gabriele Facciolo. 2018. An Analysis and Implementation of Multigrid Poisson Solvers With Verified Linear Complexity. *Image Processing On Line* 8 (2018), 192–218. https://doi.org/10.5201/ipol.2018.228

Sean Ryan Fanello, Julien Valentin, Adarsh Kowdle, Christoph Rhemann, Vladimir Tankovich, Carlo Ciliberto, Philip Davidson, and Shahram Izadi. 2017. Low Compute and Fully Parallel Computer Vision With HashMatch. In *The IEEE International Conference on Computer Vision (ICCV)*.

[8] https://developer.qualcomm.com/software/fastcv-sdk

Michael Goesele, Jens Ackermann, Simon Fuhrmann, Carsten Haubold, Ronny Klowsky, Drew Steedly, and Richard Szeliski. 2010. Ambient Point Clouds for View Interpolation. *ACM Trans. Graph.* 29, 4, Article 95 (July 2010), 6 pages. https://doi.org/10.1145/1778765.1778832

Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. 2017. Casual 3D Photography. *ACM Trans. Graph.* 36, 6, Article 234 (Nov. 2017), 15 pages. https://doi.org/10.1145/3130800.3130828

Peter Hedman and Johannes Kopf. 2018. Instant 3D Photography. *ACM Trans. Graph.* 37, 4, Article 101 (July 2018), 12 pages. https://doi.org/10.1145/3197517.3201384

Heiko Hirschmüller. 2008. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 2 (February 2008), 328–341.

H. Hirschmüller, M. Buder, and I. Ernst. 2012. Memory Efficient Semi-Global Matching. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* I3 (Jul 2012), 371–376. https://doi.org/10.5194/isprsannals-I-3-371-2012

Aleksander Holynski and Johannes Kopf. 2018. Fast Depth Densification for Occlusion-aware Augmented Reality. *ACM Trans. Graph.* 37, 6, Article 194 (Dec. 2018), 11 pages. https://doi.org/10.1145/3272127.3275083

Alexander Hornung and Leif Kobbelt. 2009. Interactive Pixel-Accurate Free Viewpoint Rendering from Images with Silhouette Aware Sampling. *Comput. Graph. Forum* 28, 8 (2009), 2090–2103. https://doi.org/10.1111/j.1467-8659.2009.01416.x

M. Jakubowski and G. Pastuszak. 2013. Block-based motion estimation algorithms — a survey. *Opto-Electronics Review* 21, 1 (01 Mar 2013), 86–102. https://doi.org/10.2478/s11772-013-0071-0

Takeo Kanade, Atsushi Yoshida, Kazuo Oda, Hiroshi Kano, and Masaya Tanaka. 1996. A Stereo Machine for Video-rate Dense Depth Mapping and Its New Applications. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*. 196–202.

Anat Levin, Dani Lischinski, and Yair Weiss. 2004. Colorization using optimization. In *Proc. SIGGRAPH*. 689–694. https://doi.org/10.1145/1186562.1015780

Christian Lipski, Christian Linz, Kai Berger, Anita Sellent, and Marcus Magnor. 2010. Virtual Video Camera: Image-Based Viewpoint Navigation Through Space and Time. *Comput. Graph. Forum* 29, 8 (2010), 2555–2568. https://doi.org/10.1111/j.1467-8659.2010.01824.x

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. In *Proceedings of High Performance Graphics*. ACM, 13.

L. McMillan and G. Bishop. 1995. Plenoptic Modeling: An Image-Based Rendering System. In *ACM SIGGRAPH 1995 Conference Proceedings*. 39–46.

Harris Nover, Supreeth Achar, and Dan Goldman. 2018. ESPReSSo: Efficient Slanted PatchMatch for Real-time Spacetime Stereo. In *International Conference on 3D Vision*.

J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. 2004. Video coding with H.264/AVC: tools, performance, and complexity. *IEEE Circuits and Systems Magazine* 4, 1 (2004), 7–28. https://doi.org/10.1109/MCAS.2004.1286980

Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson image editing. *ACM Trans. Graph. (Proc. SIGGRAPH)* 22, 3 (July 2003), 313–318. https://doi.org/10.1145/882262.882269

Christian Richardt, Douglas Orr, Ian Davies, Antonio Criminisi, and Neil A. Dodgson. 2010. Real-time Spatiotemporal Stereo Matching Using the Dual-Cross-Bilateral Grid. In *Proceedings of the European Conference on Computer Vision (ECCV) (Lecture Notes in Computer Science)*, Vol. 6313. 510–523. https://doi.org/10.1007/978-3-642-15558-1_37

Daniel Scharstein and Richard Szeliski. 2002. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision* 47, 1 (May 2002), 7–42. http://vision.middlebury.edu/stereo/

H.-Y.. Shum, S.-C. Chan, and S. B. Kang. 2007. *Image-Based Rendering*. Springer, New York, NY.

Timo Stich, Christian Linz, Georgia Albuquerque, and Marcus Magnor. 2008. View and Time Interpolation in Image Space. *Comput. Graph. Forum* 27, 7 (2008), 1781–1787.

Richard Szeliski. 2006. Locally Adapted Hierarchical Basis Preconditioning. *ACM Trans. Graph.* 25, 3 (July 2006), 1135–1143. https://doi.org/10.1145/1141911.1142005

Richard Szeliski. 2010. *Computer Vision: Algorithms and Applications* (1st ed.). Springer, New York. http://szeliski.org/Book

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 (Dec. 2018), 19 pages. https://doi.org/10.1145/3272127.3275041

Table 4. Spatial consistency checks applied on motion vectors from the video encoder, from top of bottom. Failure rates reflects percentage of candidate motion vectors input to *that* stage, not the total motion vectors.

Motion vector filter	**Description**	**Failure criteria**	**Failure %**
Encoder I-block	Failure to find a correspondence.	Internal to video encoder.	54 %
Inter-subframe motion vectors	Motion vectors must be contained within a given subframe.	Motion vector endpoint is outside the subframe.	1 %
Left / right consistency	Motion vectors from left to right ($L \rightarrow R$) must correspond to those from right to left ($R \rightarrow L$).	x component of $L \rightarrow R$ motion vector differs from matching $R \rightarrow L$ motion vector by more than 1 pixel.	22 %
Epipolar consistency	Motion vectors must be horizontal.	Absolute value of y component of motion vector is greater than 1 pixel.	12 %
Parabola fit or subpixel refinement	The parabola fit on ZSSD error values between the motion vector start point in the right image and the motion vector end point $[-1, 1]$ in the left image. The patch size corresponds to macro-block size.	No local minimum for parabola fit within $[-1, 1]$ pixel of the motion vector, or parabola fit with second order term below a given threshold.	18 %

C. Lawrence Zitnick and Sing Bing Kang. 2007. Stereo for Image-Based Rendering using Image Over-Segmentation. *Int. J. Comput. Vision* 75, 1 (Oct. 2007), 49–65. https://doi.org/10.1007/s11263-006-0018-8

C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. 2004. High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics (Proc. SIGGRAPH 2004)* 23, 3 (August 2004), 600–608.

A VIDEO ENCODER PARAMETERS

The most important video encoder parameters we tuned to improve 3D reconstruction are listed below. Ostermann et al. give a detailed discussion of parameters for H.264 encoders.

- Subframes vs. sending multiple frames. The HW video encoder latency consists of both a fixed overhead and a resolution-dependent overhead. For VGA resolution, the fixed overhead is significantly higher, and therefore, it is more efficient to send a single input with subframes to the encoder. The "MV crosstalk" between sub-frames is relatively low (1 % of the MVs) and can be filtered from the output.
- Encoder bit rate. Higher bitrate leads to fewer higher quality MVs and higher CPU decoding time. Variable bitrate produced higher quality MVs vs. constant bitrate.
- I-block period. The approach sends an I-P-P frame (left-right-left) sequence to the encoder for each input stereo pair. The P-frames are constrained such that they can only refer to the preceding frame. For longer I-block periods (e.g. (I-P-P)-(P-P-P)-(I-...)), we observed significant increases in outliers and temporal flicker.