This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2459902, IEEE Transactions on Visualization and Computer Graphics

MobileFusion: Real-time Volumetric Surface Reconstruction and Dense Tracking On Mobile Phones

Peter Ondrúška, Pushmeet Kohli and Shahram Izadi

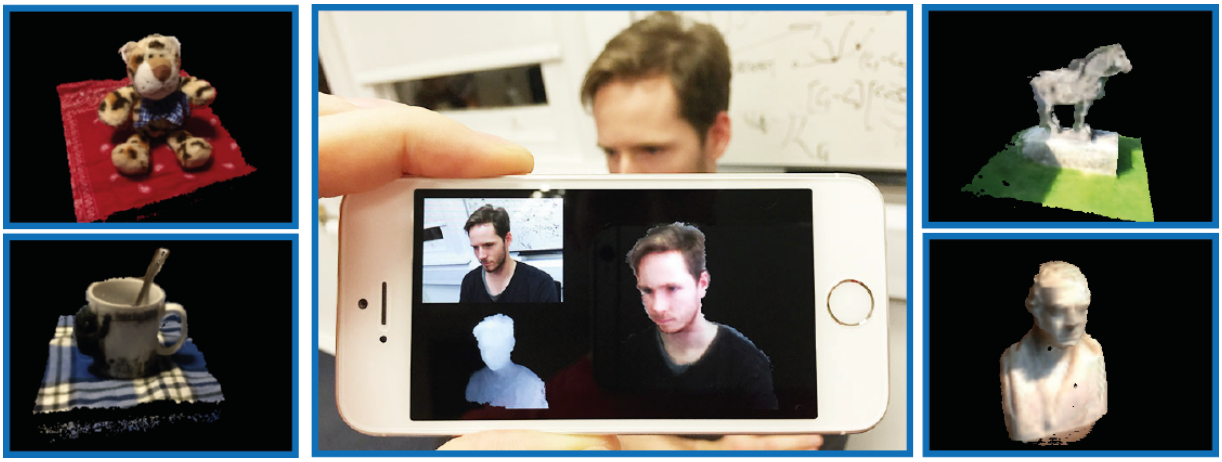

Fig. 1. Example objects scanned in real-time on a mobile phone using our system. Note we only use the internal RGB camera, and all computation is performed on the device.

Abstract—We present the first pipeline for real-time volumetric surface reconstruction and dense 6DoF camera tracking running purely on standard, off-the-shelf mobile phones. Using only the embedded RGB camera, our system allows users to scan objects of varying shape, size, and appearance in seconds, with real-time feedback during the capture process. Unlike existing state of the art methods, which produce only point-based 3D models on the phone, or require cloud-based processing, our hybrid GPU/CPU pipeline is unique in that it creates a connected 3D surface model directly on the device at 25Hz. In each frame, we perform dense 6DoF tracking, which continuously registers the RGB input to the incrementally built 3D model, minimizing a noise aware photoconsistency error metric. This is followed by efficient key-frame selection, and dense per-frame stereo matching. These depth maps are fused volumetrically using a method akin to KinectFusion, producing compelling surface models. For each frame, the implicit surface is extracted for live user feedback and pose estimation. We demonstrate scans of a variety of objects, and compare to a Kinect-based baseline, showing on average $\sim 1.5cm$ error. We qualitatively compare to a state of the art point-based mobile phone method, demonstrating an order of magnitude faster scanning times, and fully connected surface models.

Index Terms—3D object scanning, surface reconstruction, mobile computing

1 Introduction

In recent years, low-cost object scanning has become a key consumer scenario, spurred on by interest around 3D printing, and more ubiquitous ways to consume and share 3D models on our devices (e.g. WebGL). The importance of providing *live* feedback during the acquisition process is particularly important, and ensures that the object or scene is captured without missing important details or parts. Systems such as KinectFusion [16, 9] have demonstrated the importance of live capture of 3D *surface* models as opposed to point-based representations (e.g. [25]). Surfaces encode connectivity information, allowing real-world geometry to be modeled in a spatially continuous manner. These connected surface models also seamlessly integrate into existing 3D applications and tools e.g. for viewing and editing 3D content, printing or game physics. However, the majority of existing methods for live 3D surface reconstruction require *active* depth sensors and/or high end GPUs. This limits object scanning scenarios to PCs or high end laptops and tablets, with custom hardware. Whilst work, such as Google's Tango project [8] have begun to explore more integrated solutions for mobile depth camera hardware, the cost, form-factor, and power considerations have yet to make such devices ubiquitous.

We present a new system for making real-time scanning of 3D *surface* models even more cheaper and ubiquitous, using mobile phones we already have in our pockets, without any hardware modification. Existing state of the art mobile methods approximate surfaces using simple visual hull constraints [23], Delaunay triangulation [18], or purely point-based representations [28, 12]; or use cloud-based processing to avoid on-device computation [2]. Instead our system reconstructs real-time 3D *surface* models directly on the device, allowing lightweight capture of detailed 3D objects, at speeds that have yet to be demonstrated by other systems. We describe our pipeline in full, emphasizing compo-

- *Peter Ondrúška is a DPhil student at Mobile Robotics Group, University of Oxford. E-mail: ondruska@robots.ox.ac.uk.*
- *Pushmeet Kohli is at Microsoft Research. E-mail: pkohli@microsoft.com.*
- *Shahram Izadi is at Microsoft Research. E-mail: shahrami@microsoft.com.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2459902, IEEE Transactions on Visualization and Computer Graphics

nents for uncertainty-aware dense model tracking, robust key-frame selection, dense stereo, volumetric fusion and surface extraction, and their efficient mobile phone implementation, which allows for 25Hz performance. We show qualitative results of our system (see Figure 1 and accompanying video [1]), and compare to the point-based method of [28, 12]. Additionally, we compare the accuracy of our method quantitatively against a consumer depth camera baseline using a new dataset of 3D reconstructions which we make public.

Our key contributions can be summarized as:

- The first real-time fully dense *surface* reconstruction pipeline for mobile phones, which operates only on live RGB input, with computation fully on the device. This includes at 25 frames-per-second: dense camera tracking, key frame selection, dense stereo matching, volumetric depth map fusion, and raycast-based surface extraction.
- A dense, feature-free, six degree-of-freedom (6DoF) tracker, registering the RGB data to the volumetric surface model in real-time. Whilst semi-dense methods for mobile phone hardware exist [26], to our knowledge, this is the first time that a fully dense method has been demonstrated. We extend [17] modeling uncertainty directly from the implicit volumetric surface, and fast GPU-based model extraction and alignment.
- Demonstration of volumetric depth map fusion [4, 16, 9] on commodity mobile phones. Including extensions to [21] for more robust key-frame selection, and [16, 9] for more efficient depth map fusion and extraction.
- Compelling new object scanning examples, demonstrating extracted 3D meshes at speeds and quality yet to obtained with off-the-shelf mobile phone hardware.
- A new dataset of 3D models that enables quantitative evaluation of methods for dense *surface* reconstruction. [2]

2 Related Work

In terms of active sensors, early 3D scanning systems used custom structured light sensors and point-based techniques for 3D object acquisition [25, 30]. The advent of consumer depth cameras such as Kinect, and GPGPU computation capabilities gave rise to real-time *surface* reconstruction systems such as KinectFusion [16, 9], based on more sophisticated volumetric depth map fusion techniques [4]. Whilst exciting work exists on mobile depth cameras [8, 24], few mobile phones have such capabilities today. The RGB camera remains the prevalent and ubiquitous visual sensor, and due to costs, power, form-factor and photo capture scenarios, this is likely to remain the case in the near future.

The ubiquity of visible light cameras has motivated much research in real-time or live structure from motion (SfM), multi-view stereo (MVS) or simultaneous localization and mapping (SLAM). Early work demonstrated real-time disparity estimation using handheld cameras [20], as well as live [19] or efficient [32] GPU-based depth map fusion techniques. This line of research even explored extending the implicit volumetric surface reconstruction techniques proposed for active sensors [4].

Early SLAM systems [11, 5] instead looked at *sparse* mapping using single monocular cameras, for the purpose of real-time 6DoF tracking. More recently, there has been a shift towards semi-dense [6, 7] or fully dense [14] tracking techniques but again the purpose has remained robust camera tracking rather than detailed 3D reconstruction per say. Dense stereo matching techniques have been combined with either sparse [21, 15, 27] or dense trackers [14] to create compelling surface reconstructions. In this later work [21, 14] per frame, dense stereo computation was combined with volumetric techniques [4, 16, 9].

[1] Video is available at: `http://youtu.be/5tsLLq02xnk`

[2] Datasets are available at: `http://mrg.robots.ox.ac.uk/mrg_people/peter-ondruska/`

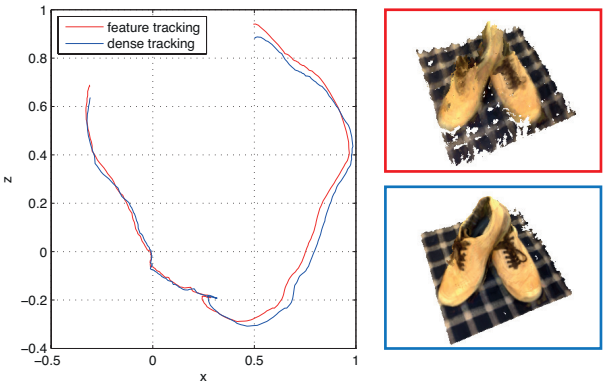

Fig. 3. Camera pose estimation based on feature-based tracking (red) and dense tracking (blue) together with resulting models. Dense tracking increases robustness in situations without salient image features visible at all times.

However, these systems require heavyweight GPGPU pipelines, which even with the new generation of mobile graphics hardware cannot be deployed on mobile devices without major adaption.

The technical challenges of mobile reconstruction combined with the ubiquity of such devices has led to much interest in this space. Certain approaches simply avoid the computation problem by offloading to servers [31] or the cloud [2], but this requires continual high bandwidth connectivity, and often reconstructions can take many minutes to be completed [2]. SLAM systems have explored efficient 6DoF pose estimation on mobile devices [29, 10, 26, 24], but have avoided the challenge of reconstructing detailed surfaces under such computational constraints. [18] use panoramic images captured on a mobile device to reconstruct building facades, but can only coarsely approximate 3D geometry using Delaunay-based triangulation. [23] reconstructs convex 3D shapes in real-time on a mobile phone, using a silhouette-based multi-view approach. [28, 12] create point-based 3D models using a mobile phone, but lack the ability of reconstructing connected surface models, as shown in their supplementary video. Further, their approach takes several minutes to scan objects.

Our approach differs to these techniques in that we support full volumetric surface reconstruction and dense tracking in real-time on the phone. To the best of our knowledge this is the first time that dense detailed surfaces have been captured with off-the-shelf mobile phone hardware. Interestingly, even custom depth camera mobile phones such as Google's Tango have yet to support such surface-based reconstructions, relying on point-based techniques only.

3 System Overview

Our pipeline is shown in Figure 2, and comprises of five main steps: dense image alignment for pose estimation, key frame selection, dense stereo matching, volumetric update, and raytracing for surface extraction. Given a stream of RGB image frames coming from the camera, the tracker estimates the 6DoF camera pose by aligning the entire RGB frame with a projection of the current volumetric model. Key-frames are selected from this live input, using an efficient and robust metric that evaluates the overlap between frames. Next, the camera pose is used to select a key-frame which after rectification can along with the live frame be used to perform dense stereo matching. The output depth map is then fused into a single global volumetric model. While scanning the visualization of the current model from camera perspective is displayed to the user for live feedback, and then passed into the camera tracking phase for pose estimation.

As depicted in Figure 2, we use I_i to denote the camera image frame at time step i. The 6DoF pose of the camera corresponding to this image will be denoted by matrix T_i (encoding a 3D rotation and translation). We use a volumetric model [4] for representing the surface model, where a truncated signed distance function (TSDF) is stored in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2459902, IEEE Transactions on Visualization and Computer Graphics

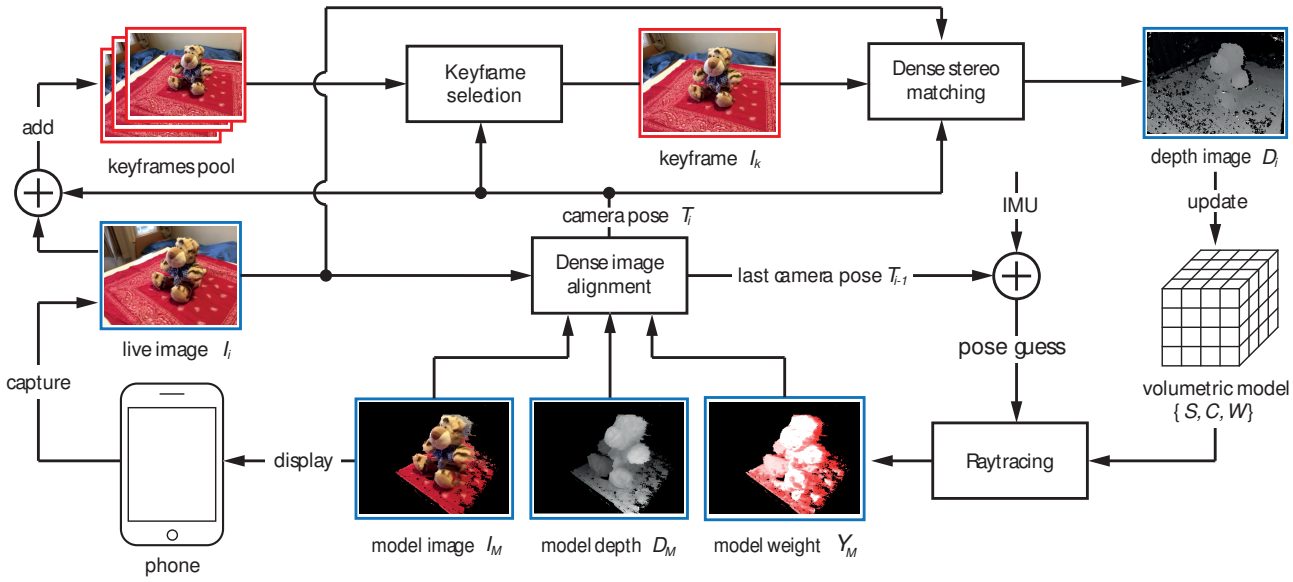

Fig. 2. System pipeline. See text for details.

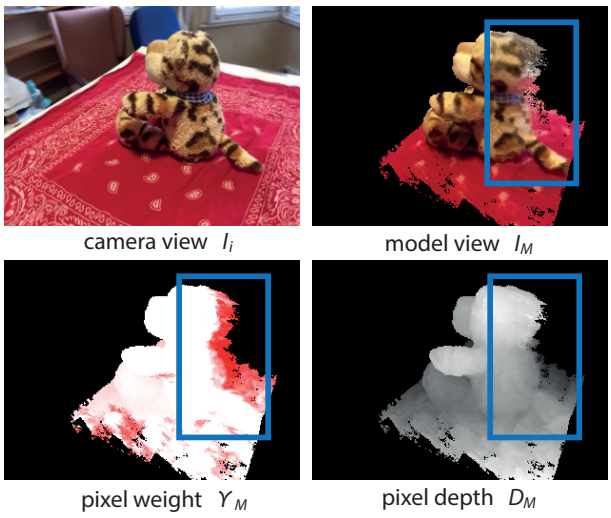

Fig. 4. Detail of the weighted image alignment scheme for camera tracking. Model view $\{I_M, D_M\}$ is used for per-pixel alignment of camera image I_i. We downweight pixels corresponding to noisy parts of the model using the projection Υ_M of weights $W(p)$ stored in the TSDF.

voxel grid. This volumetric model is represented by a voxel grid P of size $K \times K \times K$. Each voxel $p \in P$ is associated with three quantities: the value of the TSDF $S(p)$ which encodes the signed distance of the voxel to the nearest surface measurement, the color vector $C(p)$ and the cumulative weight $W(p)$ that represents the confidence the algorithm has in the TSDF and color values corresponding to voxel p.

4 Uncertainty-Aware Camera Pose Estimation

Estimation of the camera pose plays a critical role in our scanning system and significantly affects the quality of the resulting model. Achieving high tracking performance is particularly challenging in our 'outside-in' scanning scenarios, as the camera observes very different views as it moves around the object. We observed that using a feature-based camera tracker often results in position drift or even tracking failure as a result of the lack of robust global features that are visible in all views. This drift affects the performance of all down-stream stages of the pipeline and leads to poor reconstruction (see Figure 3).

To make tracking robust, we use a dense *feature-free* monocular camera tracking approach, which has been shown superior to feature-based methods [17, 26]. In our approach, the camera pose is being estimated directly using the volumetric model being built. Using this dense tracking pipeline, the pose of the camera is computed based on the per-pixel photo-consistency of the rendered model and the live camera image. In general we seek a pose T_i from the incoming camera input image I_i where the rendered image of the (partially completed) model aligns with the observed input image. The pose is computed using a 2.5D image alignment [17]. Let I_M be a projection of the model $\{S,C,W\}$ from pose guess T_v close to true camera pose and D_M is the corresponding depth of every pixel $u \in \Omega$ where Ω is the set of valid pixels of the rendered model. We estimate pose T_v from the last camera frame T_{i-1} optionally updated by incorporating IMU information. Relative pose T_{vl} of incoming frame I_i is obtained by minimizing the photometric error

$$F(T_{vl}) = \sum_{u \in \Omega} (f_u(T_{vl}))^2 \quad (1)$$

$$f_u(T_{vl}) = I_i(\alpha(T_{vl}, u, D_M(u))) - I_M(u) \quad (2)$$

where $\alpha(T_{vl}, u, d)$ is a projection of a point (u, d) from rendered model view I_M into camera image I_i given by transform T_{vl}. The final pose T_i is then given as $T_i = T_{vl} T_v$. Using this method the optimal pose is computed robustly based on a dense consensus of pixels, instead of sparese features used in tracking methods such as [10]. Note this 2.5D optimisation scheme requires only colour I_M and depth D_M of the rendered model, and only the camera image I_M. Specifically it does not require input depth of pixels in the camera image D_i. This depth is instead computed separately at a later stage of the pipeline, after the pose T_i is known as described in Section 5.

Projecting the Model Model projection for given camera pose T_v is achieved through raycasting in a manner similar to [16, 9]. Specifically, for each pixel u, we march along the corresponding ray in the volumetric grid P until a zero crossing in the $S(p)$ is found, which specifies the pixel depth $D_M(u)$. Pixel color is computed by interpolating $C(p)$ at this point. As the projection is one of the most computationally demanding parts of algorithm, the efficient implementation is more closely explained in Section 7.

Noise aware photometric error While the dense model based tracking (as defined above) works well, it can still lose tracking in certain conditions. We observed that a major source of these errors was the presence of erroneous depth values that had been aggregated in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2459902, IEEE Transactions on Visualization and Computer Graphics

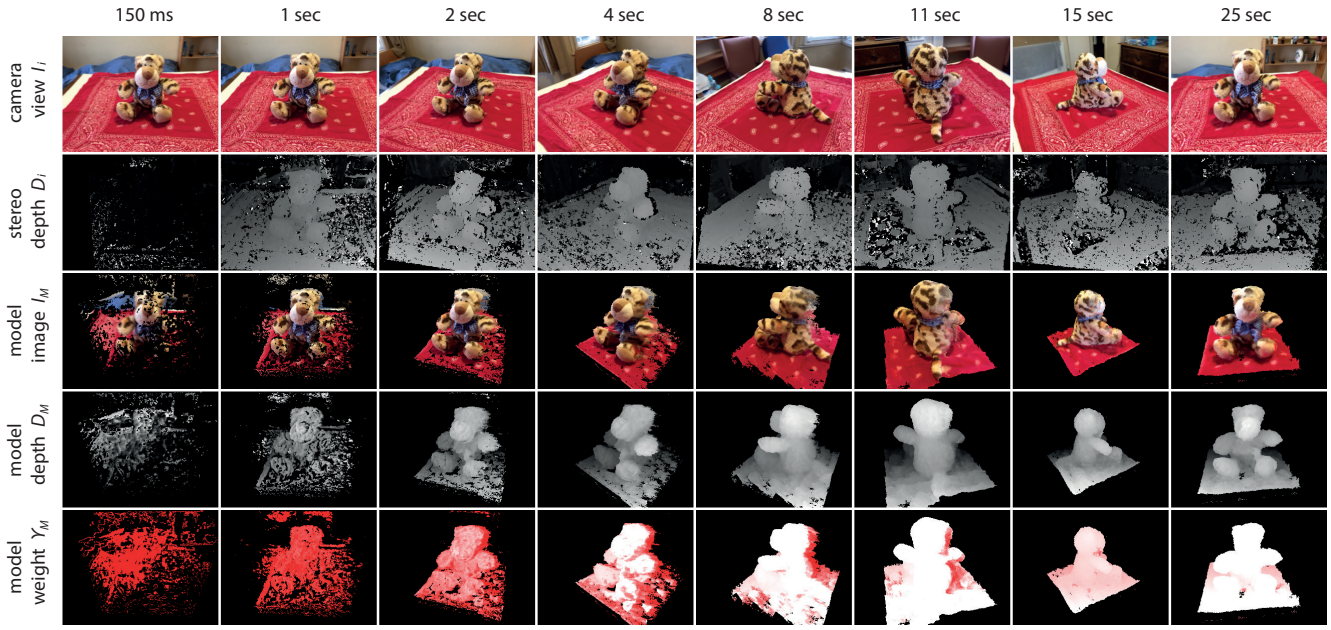

Fig. 5. Progress of object scanning from a sequence of length 25 seconds. Each column captures live camera view, per frame depth map, raytraced RGB, 3D model and model confidence, over time.

volumetric TSDF (see Figure 4) resulting in outliers in the cost function Equation 1. This is especially the case when occluded parts of the object are seen for the first time. Due to limited number of observations of these parts of the object, the volumetric data is noisy. This problem can be solved using a robust cost function based on the Huber norm. Using the Huber norm however makes the optimisation more complicated as it involves iterative re-weighting Levenberg-Marquardt minimization [26]. Instead we used a simpler method which explicitly down-weights contribution of pixels in I_M originating from recently initialised voxels i.e. having small voxel weight $W(p)$. Specifically we assign a weight to each pixel u such that the optimized photometric error becomes

$$F(T_{vl}) = \sum_{u \in \Omega} \Upsilon_M(u)(f_u(T_{vl}))^2. \quad (3)$$

Weights Υ_M for each pixel u are computed simply by projecting voxel weight function $W(p)$ which in turn is computed using the number and value of observations that have been made for each voxel. Minimization of the uncertainty aware photometric error defined above is achieved through simple Gauss-Newton gradient descent [3, 6, 17]. In practice this optimization scheme works well for smooth camera trajectories and video frame rates (25Hz). It does not even require a coarse-to-fine pyramidal approach [26, 27]. For rapid motion, a more efficient pyramidal approach together with a more robust cost function can be used to further improve tracking performance.

Figure 6, shows progress of minimizing the cost function in Equation 3. In this plot we see the importance of frame-rate for fast convergence of our optimizer. In our system we maintain 25Hz for the full pipeline, which given the smaller distances traveled per frame results in convergence after around 3 frames. As shown, if our system was slower then the number of solver iterations would be significantly higher.

5 Monocular Depth Estimation

After pose estimation, we estimate depth D_i of the input image using a simple pair-wise stereo matching method. Using the current camera image and one selected key-frame from the past, this method traingulates the depth of every pixel using a block-matching schema to find dense pixel correspondances between both frames. We chose this method as opposed to methods using multiple-frame stereo [27] as it allows faster incremental build-up of the model, and in practice, the level of redundancy of per-frame estimation allows filtering of depth outliers

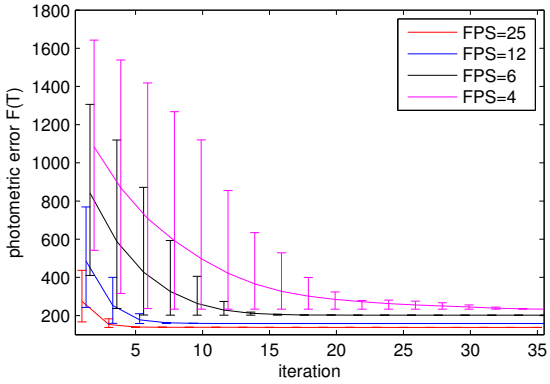

Fig. 6. Iterative alignment of live image during camera tracking. Our plots show the importance of maintaining high frame-rate camera tracking to ensure fast and precise convergence.

caused by occlusions or lack of texture. Moreover as further described in Section 7. the block-matching can be implemented extremely efficiently on a mobile phone GPU. This allows for a fast depth-estimation system that runs at 25Hz as opposed to 0.5Hz of [28, 12] coarse-to-fine scheme.

Key-frame Selection We maintain a pool of last $M = 40$ RGB frames and choose appropriate key-frame I_k for stereo matching, based on very simple but robust scoring function which can be computed extremely efficiently given just knowledge of camera poses. Upon receiving a new input frame, we look at this pool to select a key-frame. We need to trade-off between two objectives during key-frame selection. Firstly, to ensure that depth can be computed for as many pixels as possible in the image, we want to ensure that there is a high-overlap between the frame and the current camera view. In other words, the key-frame captures the same part of the scene as the current camera view. This objective can be quantified in terms of the fraction of overlap $S_\cap(T_k)$ between the rectified camera image and considered key-frame.

Secondly, we need to make sure that the stereo pair containing the key-frame has an appropriate baseline. Having a very small baseline would mean that depth for objects that are farther away cannot be

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2459902, IEEE Transactions on Visualization and Computer Graphics

computed. Here, there is a trade-off between the width of the baseline, accuracy and maximum disparity search range, where the depth of objects less than $1/d_{max}$ cannot be estimated. As the scanned object is inside the voxel grid, projecting the 8 vertices of the voxel cube provide upper $\delta_{max}(T_k)$ and lower bound $\delta_{min}(T_k)$ for the object disparity. To penalize either disparity of pixels being over the designated range or utilizing only a small fraction of it, we consider the term $S_d(T_k) = -\left|1 - \frac{\delta_{max}(T_k)}{d_{max}}\right|$ and the final scoring function is simply a sum of these two terms:

$$S(T_k) = S_{\cap}(T_k) + S_d(T_k) \tag{4}$$

Disparity Estimation The depth map for each frame of the image stream can be computed through dense stereo matching between the rectified pair of current camera view (I_i,T_i) and past key-frame (I_k, T_k) in a manner similar to [21]. An interesting trade-off is working with a simpler and faster, but less accurate, stereo method, as a opposed to one that is more precise but less efficient. In practice, given other parts of the pipeline (in particular camera tracking) rely on maintaining high framerates, and given our ability to fuse every frame volumetrically. We found that going for efficiency and less per frame accuracy, but with high amount of redundancy in terms of computed and fused depths maps leads to compelling 3D reconstructions.

However, as highlighted, it is critical to maintain framerate for robust tracking (and hence reconstruction), therefore instead of using a propagation-based stereo method, mobile device reduces a number of computational constraints. These considerations prevented us from using sophisticated but slow algorithms for stereo matching as they lead to drastic reductions in the frame rate and result in poor tracking and 3D reconstruction.

This trade-off led us to a simple but fast, block matching approach for disparity estimation. As further discussed in Section 7, we implemented the block matching method to run on the mobile phone GPU which allowed us to perform stereo matching in real-time. We performed block matching over $d_{max} = 64$ steps along the epipolar line with patch size of 5x5 pixels, and sum of square error matching score. We also experimented with larger patch sizes which resulted in smoother but less detailed depth maps. We do L-R check to remove outliers caused by occlusions or lack of texture. Furthermore we obtained subpixel accuracy by post-processing the depth maps using parabola fitting in the neighbourhood of the disparity with minimum cost. Implementation of these operation on the GPU allowed us to generate depth maps of 320x240 resolution at 25Hz.

Note we do not provide any further post-processing refinement of depth-maps such as total-variation optimisation [27] or spatial-regularisation [26] as our aim is not to generate high-quality individual depth maps but rather fuse many moderate-quality depth maps with regularisation inherently implemented in the volumetric fusion step.

6 Fusing Depth Images

For every generated depth map D_k, we fuse the observations to refine our volumetric model. This step is akin to the voxel update performed in [16], but as described later, can be performed for all voxels independently without requiring an explicit sweep of the volume. Using volumetric fusion has numerous advantages including robust handling of outliers and efficient model update. Moreover as opposed to unstructured point cloud or surfel model representations used in recent mobile approaches [12, 28], this volumetric approach provides a continuous and connected implicit *surface* model, which can be extracted using raycasting or marching cubes [13].

For any voxel $p \in P$, given it's projection into camera space $q = T_i p$, and the projection into camera view $\pi(q) = (f_x \frac{q_x}{q_z} + c_x, f_y \frac{q_y}{q_z} + c_y)$, we compute the quantities:

$$s_i(p) = D_i(\pi(q)) - q_z \tag{5}$$
$$c_i(p) = I_i(\pi(q)) \tag{6}$$

Fig. 7. Effect of different model resolution and FPS on model quality. Our current implementation supports $K = 128$ at 25Hz and $K = 256$ at 15Hz. Note that processing speed is however the more dominant contributor to model quality. This is because at higher frame-rates, the redundancy of depth maps fused volumetrically leads to greater denoised results even at lower resolutions.

and use them to perform a weighted update of the model as

$$S_i(p) = \frac{S_{i-1}(p) \cdot W_{i-1}(p) + s_i(p) \cdot w_i(p)}{W_{i-1}(p) + w_i(p)} \tag{7}$$
$$C_i(p) = \frac{C_{i-1}(p) \cdot W_{i-1}(p) + c_i(p) \cdot w_i(p)}{W_{i-1}(p) + w_i(p)} \tag{8}$$
$$W_i(p) = W_{i-1}(p) + w_i(p) \tag{9}$$

where $w_k(p)$ is the weight of the new measurement. Note we only update voxels which project inside the camera image and for which the pixel depth information is valid (i.e. was not rejected by the L-R check of stereo depth computation step as an outlier). Filtering out depth measurements outliers greatly reduces the amount of noise in the depth fusion step. For this reason we have used a constant weight $w_k(p) = 1$ for new measurements and found no considerable difference with weighted average methods. Moreover as in [16] we only update voxels within a range around the surface having $s_k(p) > -\mu$ (i.e. visible voxels) and truncate the remaining $s_k(p)$ to $[-\mu, \mu]$ (where μ = 8 voxels). Similarly, we limit the maximum weight $W_k(p)$ to $W_{max} = 50$. The voxel resolution and frame-rate are again an important trade-off for reconstruction quality, as shown in Figure 7.

7 Mobile Phone Implementation

In this section we describe some of the important factors which allowed a real-time implementation of the described pipeline on an Apple iPhone 6. Our implementation utilizes both CPU and GPU of the mobile phone to process the camera stream at a resolution of 320x240 at 25Hz for both tracking and reconstruction.

Initialization To bootstrap the system we use a publicly available feature-based tracker [1], although open source trackers such as [10] or [26] can be used instead. Running this tracker, the user first interactively specifies the position of the volumetric grid (see accompanying video), such that the desired object is placed within. The tracker is used to provide pose T_i to initialize the model and dense tracking pipelines.

Mobile CPU & GPU Utilization Despite significant development of mobile devices in recent years, they are still 5-20x less powerful compared to their desktop counterparts. We measured the raw peak CPU performance of our platform to be 3GFlops using SIMD instructions and 70GFlops on GPU using OpenGL ES 2.0 shaders. Where possible we chose to use the GPU for demanding parallelizable tasks

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2459902, IEEE Transactions on Visualization and Computer Graphics

Operation	Unit	Running time (ms) 5	5s	6
Camera alignment	CPU	38.3	34.5	32.6
Key-frame selection	CPU	5.3	4.5	4.1
Stereo depth computation	GPU	67.1	18.4	17.3
Model update (K=128)	GPU	21.3	8.5	8.1
Model update (K=256)	GPU	68.8	26.1	25.4
Model raycasting (K=128)	GPU	46.7	16.2	15.3
Model raycasting (K=256)	GPU	81.8	26.3	24.2

Table 1. Running time of different stages of pipeline on iPhone 5, 5s and 6. Our method spreads computation across CPU and GPU to parallelize further. The CPU is used for tracking on the current input frame, whilst simultaneously the depth estimation and fusion is performed on the previous frame using GPU.

Sequence	Scanning time [sec]	Hausdorff Distance [cm]	RMS [cm]
Shoes	15	1.5821	2.0092
Teddy	25	1.1875	1.4507
Pineapple	12	1.6945	2.0000
Kangaroo	33	1.4362	1.7836
Mask	6	1.7934	2.1312

Table 2. Reconstruction error (measured by Hausdorff Distance and RMS) for different objects compared to depth camera reconstruction obtained using KinectFusion [16]. We computed both relative error with respect to voxel grid size and the corresponding absolute error in cm.

and the CPU for the tasks with sequential computation. Computational speed for each of the five main stages of our pipeline, running on three different phone models is summarized in Table 1.

CPU implementation We use CPU for both keyframe selection and dense camera alignment. CPU is especially suitable for the later as the camera alignment requires accumulation of errors across the entire input image. Utilisation of SIMD (NEON) instructions led to significant speed-ups allowing us to process 4 pixels at a time (c.f. [26]). In addition, the CPU and GPU tasks were run in parallel - camera tracking of a new frame is performed at the same time as depth computation and model update for the last frame.

GPU implementation We used GPU for stereo depth computation, model update and raycasting. We chose OpenGL ES 2.0 shaders as our GPU programming platform, due to its ubiquity on all mobile platforms. Each of these tasks can be written in the form of simple shader operating on data stored as RGBA values in textures. These textures are used to store camera images, depth maps and the full TSDF. For example, to store TSDF with $K = 256 \times 256 \times 256$, we generate a texture of size 4096×4096 with every pixel containing value for single voxel.

The use of the GPU provides three key advatanges.

First, the GPU allows effective parallelism of tasks with little or no computational dependency. For *stereo matching*, each shader independently computes depth for every pixel from a pair of rectified images. The lack of computational dependency in the block-matching scheme makes this operation very fast. Similarly *model update* through depth map fusion is carried out independently for each voxel at a time, using multiple textures. We observed that even for a volume of $K = 256$, where the model contains $8\times$ more voxels than for $K = 128$, the computational time is only slightly higher.

Next, costly operations like bi-linear interpolation of image value is efficiently implemented in hardware through texture lookup operations. This is very important for both model update and model raycasting which are heavily depend on this type of interpolation. In the model update step, the computed depth-map is repeatedly interpolated for every voxel and in the model raycasting the SDF is interpolated for every pixel.

Finally, GPU architectures natively supports 4-way SIMD parallelism by applying the same operation at every element of a RGBA vector. This is used during stereo computation (described in Section 5) where the matching cost can be simultaneously computed for 4 different disparities at the same time. This is achieved by preprocessing the image to store gray values of the pixel and three neighbouring pixels corresponding to the disparities $d+1$,$d+2$,$d+3$ as a single RGBA vector. SIMD parallelism is also utilised in both model update and raycasting when values of colour C_i and sdf S_i can be updated or interpolated at the same time.

Additionally, for model raycasting we used a special optimisation. The *raycasted* model is computed per pixel, by marching rays through the voxel grid until passing a zero crossing. Independent marching of every ray by fixed distance however results in poor performance as it exhibits random memory access. Moreover to obtain the precise point at the zero crossing a tri-linear interpolation of 8 neighboring voxels must be performed. To improve the performance of raycasting, we process all rays on a per slice basis (see Figure 8). This operation is very efficient as it involves hardware accelerated bi-linear interpolation and all the memory access is coalesced within the given texture subregion. We render each slice in order as a polygon, eliminating random memory access, and filter between slices using a custom blend operation – a task that graphics cards are extremely efficient at.

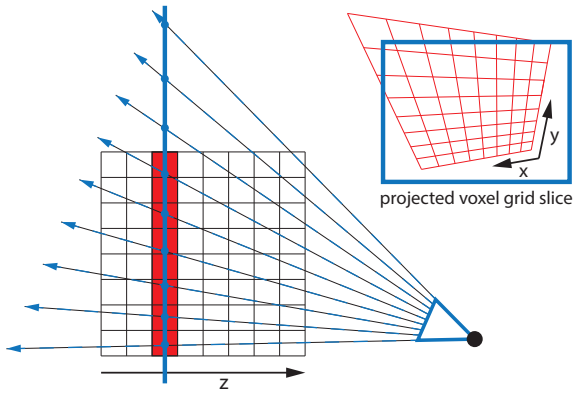

Fig. 8. Raytracing of the volumetric model minimizing random memory access. Instead of marching each ray independently, all rays are marched synchronously along an axis-parallel plane such that only one slice of the voxel grid is accessed at once.

8 Evaluation

In this section and accompanying video, we show both qualitative and quantitative results. Figure 7 shows a variety of different objects scanned which were captured in roughly 20 seconds with our method. Note the completeness of obtained surfaces, which would be challenging to achieve with point-based methods. The objects are also of varying sizes, ranging from 5^3 to 0.3^3 meters, and varying amounts of texture and appearance cues. While scanning, the user moves the phone around the entire object. We kept ISO sensitivity, exposure mode and camera focus constant to prevent changes in appearance and camera geometry caused by refocusing. The extracted colored and regular surface geometry is shown directly as extracted and rendered on the phone, at real-time rates. The bottom row of images, shows the final 3D models registered to baseline models captured with a Kinect camera, and high resolution KinectFusion implementation [22]. Table 2 highlights an average error close to 1.5cm between the two models, which is encouraging given that our system is not metric, and uses no active illumination.

Another important property of our system is the speed at which the 3D models are acquired and the continual per-frame incremental build up and preview of the 3D model. As shown in Figure 5 and supplementary video, the surface models are acquired extremely quickly, and rapidly build up live in front of the user. In Figure 11, we see a direct

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2459902, IEEE Transactions on Visualization and Computer Graphics

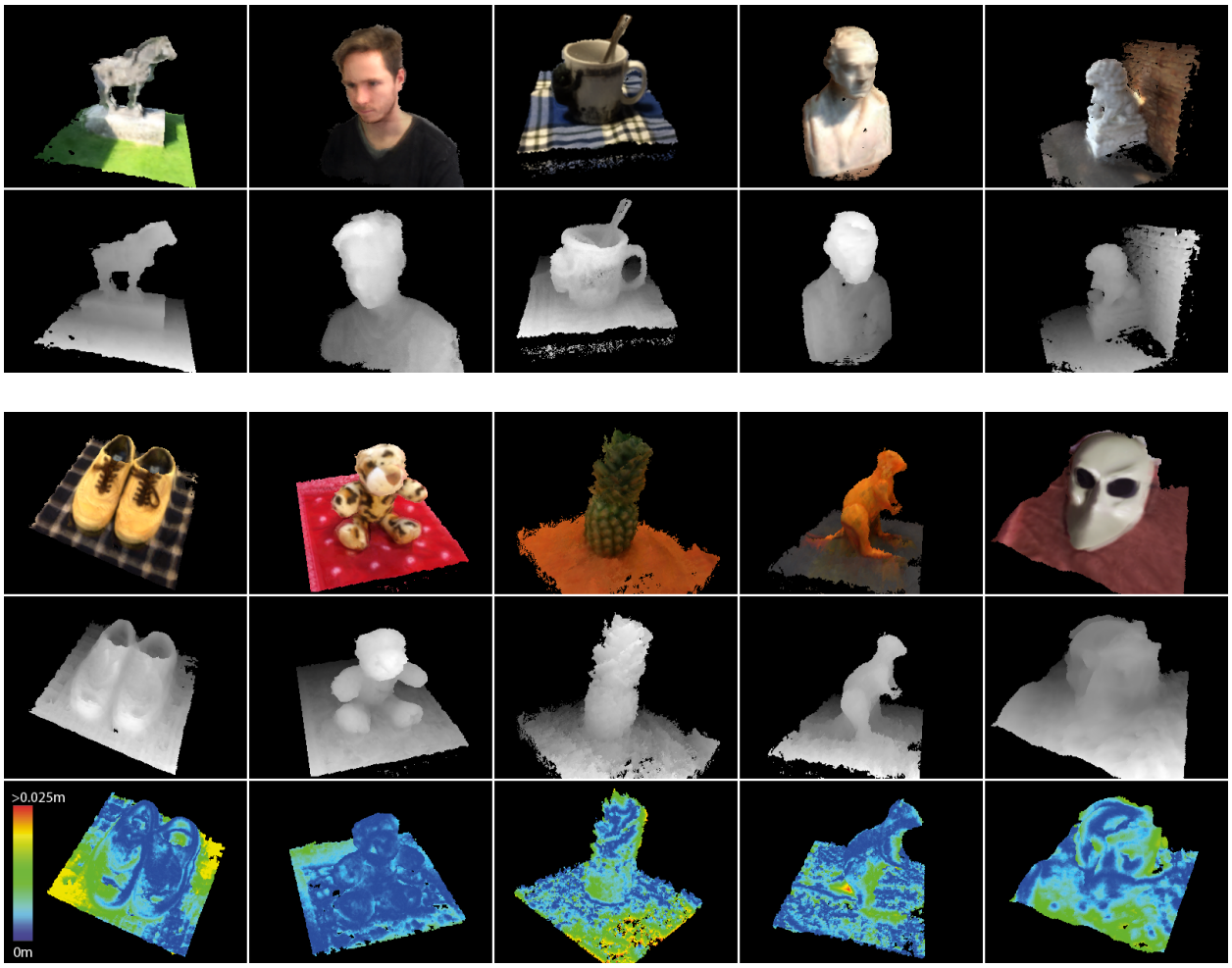

Fig. 9. Examples of generated models as object shape [bottom row] and textured shape [top row]. Each model was created during approximately 20 seconds of scanning on a mobile phone.

qualitative comparison with the method of [12]. Note that this work is a derivative of [28] with improved accuracy and efficiency.

Whilst both systems are compelling, there are clear holes in the final 3D model produced from [12] due to the point-based nature of their work. Note that our results clearly show a connected surface model. Our method can also allow the model to be acquired more quickly due to our ability to process both tracking and fusion parts of the pipeline at 25Hz. This leads to a complete capture in a matter of seconds instead of minutes.

Limitations Obviously our method also has some limitations. As with many passive stereo methods, objects with no texture cause problems. This could be fixed by using depth estimation algorithms that employ regularization to extrapolate depth values for pixels where left-right check fails. In the presence of specular or glossy surfaces, the model can contain missing geometry or noise. Color bleeding across voxels is also an issue, Figure 10. Finally, by registering continually with the accumulated 3D model, some drift can be mitigated during loop closure, but there are cases where inherent sensor drift causes more significant errors. Note also that currently we do not support camera relocalization after tracking loss. Further, the voxel resolution is currently limited due to GPU hardware, and our models are not meteric i.e. (although this can be solved using the IMU [28]).

9 Conclusions

We have presented the first pipeline for real-time volumetric surface reconstruction and dense 6DoF camera tracking running purely on standard, off-the-shelf mobile phones. Using only the embedded RGB camera, our system allows users to scan objects of varying shape, size, and appearance in seconds, with real-time feedback during the capture process. Unlike existing state of the art methods, which produce only point-based 3D models on the phone, or require cloud-based processing, our hybrid GPU/CPU pipeline is unique in that it creates a connected 3D surface model directly on the device at 25Hz. We believe these types of 3D scanning technologies, will bring 3D acquisition to a new audience, where lightweight capture of the real-world becomes truly ubiquitous.

References

[1] 13thLab. Pointcloud sdk. `http://developer.pointcloud.io/`, 2014.
[2] Autodesk. 123d app. `http://www.123dapp.com/`, 2014.
[3] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.
[4] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312, 1996.
[5] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-Time Single Camera SLAM. *PAMI*, 29(6), 2007.
[6] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.
[7] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast Semi-Direct Monocular Visual Odometry. In *ICRA*, 2014.
[8] Google. Project tango. `https://www.google.com/atap/projecttango/`, 2014.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2015.2459902, IEEE Transactions on Visualization and Computer Graphics

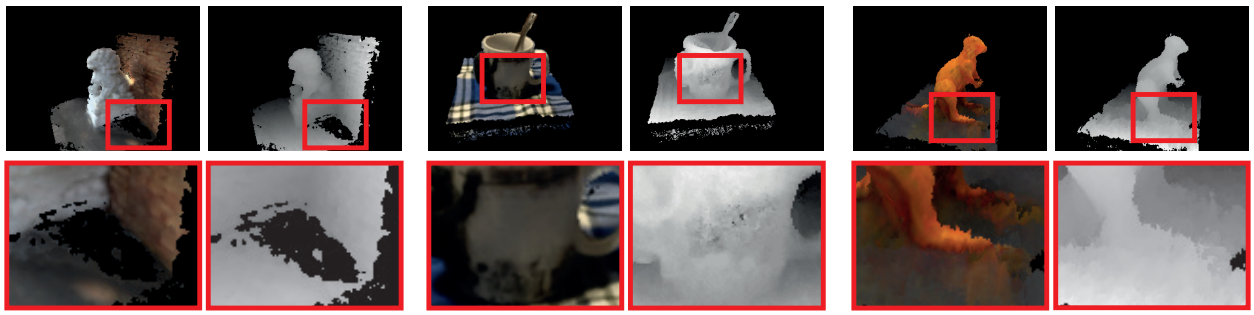

Fig. 10. Examples of some of the method limitations. In the presence of specular or glossy surfaces or textureless regions, the model can contain missing geometry [left] or noise [middle]. Color bleeding across voxels is also an issue [right].

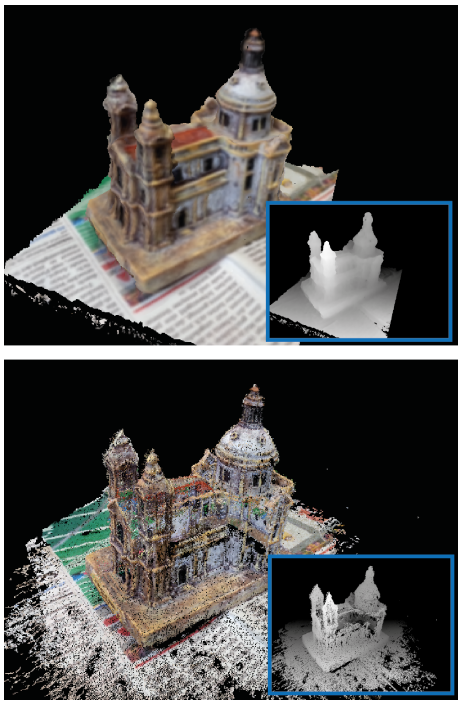

Fig. 11. Reconstruction of an object by our method [top] and point cloud-based method of [12] [bottom].

[9] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *UIST*, 2011.

[10] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE, 2009.

[11] G. Klein and D. W. Murray. Parallel tracking and mapping for small ar workspaces. In *ISMAR*, 2007.

[12] K. Kolev, P. Tanskanen, P. Speciale, and M. Pollefeys. Turning mobile phones into 3d scanners. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 3946–3953. IEEE, 2014.

[13] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[14] R. A. Newcombe. Dense visual slam. In *PhD Thesis, Imperial College*, 2012.

[15] R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1498–1505. IEEE, 2010.

[16] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *ISMAR*, 2011.

[17] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011.

[18] Q. Pan, C. Arth, G. Reitmayr, E. Rosten, and T. Drummond. Rapid scene reconstruction on mobile phones from panoramic images. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 55–64. IEEE, 2011.

[19] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, et al. Detailed real-time urban 3d reconstruction from video. *International Journal of Computer Vision*, 78(2-3):143–167, 2008.

[20] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 59(3):207–232, 2004.

[21] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche. Monofusion: Real-time 3d reconstruction of small scenes with a single web camera. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pages 83–88. IEEE, 2013.

[22] V. A. Prisacariu, O. Kähler, M. M. Cheng, C. Y. Ren, J. Valentin, P. H. S. Torr, I. D. Reid, and D. W. Murray. A Framework for the Volumetric Integration of Depth Images. *ArXiv e-prints*, 2014.

[23] V. A. Prisacariu, O. Kahler, D. W. Murray, and I. D. Reid. Simultaneous 3d tracking and reconstruction on a mobile phone. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pages 89–98. IEEE, 2013.

[24] Qualcomm. Vuforia. `http://www.vuforia.com/`, 2014.

[25] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3d model acquisition. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 438–446. ACM, 2002.

[26] T. Schops, J. Enge, and D. Cremers. Semi-dense visual odometry for ar on a smartphone. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 145–150. IEEE, 2014.

[27] J. Stühmer, S. Gumhold, and D. Cremers. Real-time dense geometry from a handheld camera. In *Pattern Recognition*, pages 11–20. Springer, 2010.

[28] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys. Live Metric 3D Reconstruction on Mobile Phones. In *ICCV*, 2013.

[29] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 125–134. IEEE Computer Society, 2008.

[30] T. Weise, T. Wismer, B. Leibe, and L. Van Gool. In-hand scanning with online loop closure. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1630–1637. IEEE, 2009.

[31] A. Wendel, M. Maurer, G. Graber, T. Pock, and H. Bischof. Dense reconstruction on-the-fly. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1450–1457. IEEE, 2012.

[32] C. Zach. Fast and high quality fusion of depth maps. In *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, volume 1, 2008.