MonoFusion: Real-time 3D Reconstruction of Small Scenes with a Single Web Camera

Vivek Pradeep* Christoph Rhemann Shahram Izadi Christopher Zach Michael Bleyer Steven Bathiche

Microsoft Corporation and Microsoft Research, Cambridge, UK

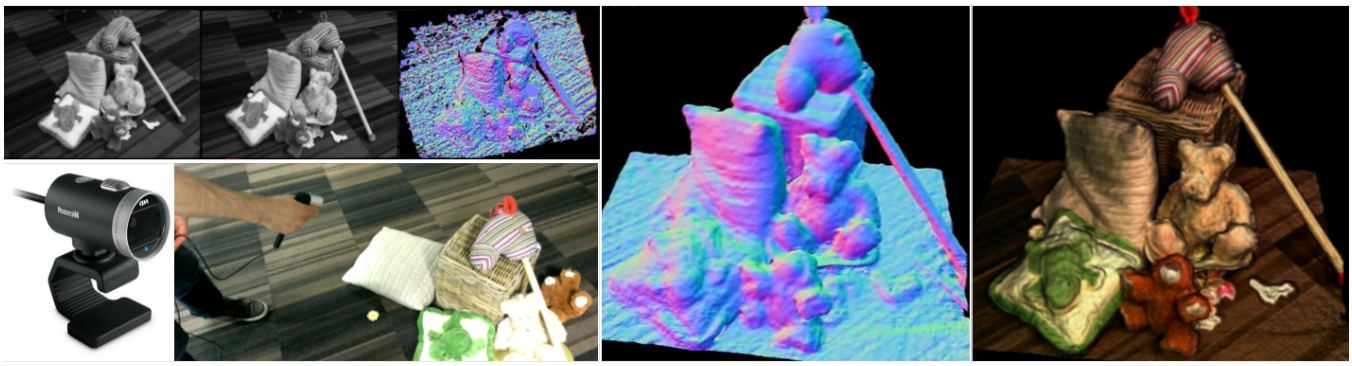

Figure 1: We present a new system for real-time 3D reconstruction using a single moving off-the-shelf web camera. Our system first estimates the pose of the camera using a sparse feature tracker. We then perform efficient variable-baseline stereo matching between the live frame and a previously selected key frame. Our stereo matcher creates a dense depth map per frame, which is then fused volumetrically into a single implicit surface representation. Our method does not require a cost volume for intermediate construction of depth maps, and instead integrates every frame directly into the voxel grid using a computationally simple fusion method. This allows cheap and readily-available web cameras to be used for small sized reconstructions for AR applications.

Abstract

MonoFusion allows a user to build dense 3D reconstructions of their environment in real-time, utilizing only a single, off-the-shelf web camera as the input sensor. The camera could be one already available in a tablet, phone, or a standalone device. No additional input hardware is required. This removes the need for power intensive active sensors that do not work robustly in natural outdoor lighting. Using the input stream of the camera we first estimate the 6DoF camera pose using a sparse tracking method. These poses are then used for efficient dense stereo matching between the input frame and a key frame (extracted previously). The resulting dense depth maps are directly fused into a voxel-based implicit model (using a computationally inexpensive method) and surfaces are extracted per frame. The system is able to recover from tracking failures as well as filter out geometrically inconsistent noise from the 3D reconstruction. Our method is both simple to implement and efficient, making such systems even more accessible. This paper details the algorithmic components that make up our system and a GPU implementation of our approach. Qualitative results demonstrate high quality reconstructions even visually comparable to active depth sensor-based systems such as KinectFusion.

1 Introduction and Background

Whilst 3D reconstruction is an established field in computer vision and graphics, it is now gaining newfound momentum due to the rise of consumer depth cameras (such as the Microsoft Kinect and Asus Xtion). Since these sensors are capable of delivering depth maps at real-time rates, a particular focus of recent systems is to perform *online* surface reconstruction. The ability to obtain reconstructions in *real time* opens up various interactive applications including: augmented reality (AR) where real-world geometry can be fused with 3D graphics and rendered live to the user; autonomous guidance for robots to reconstruct and respond rapidly to their environment; or even to provide immediate feedback to users during 3D scanning.

Many recent online systems [11, 20, 8, 10, 31, 33] have begun to embrace active depth cameras, such as the Kinect. Some [11, 20] have adopted the volumetric fusion method of Curless and Levoy [4]. This approach supports incremental updates, exploits redundant samples, make no topological assumptions, approximates sensor uncertainty, and fusion simply becomes the weighted average of existing and new samples. For active sensors, this type of fusion has demonstrated very compelling results [4, 14, 11].

Whilst active sensors have many strengths, there are certain scenarios where standard passive RGB cameras are preferred due to power consumption, outdoor use and form-factor. This has led many researchers to investigate methods that try to reconstruct scenes using only passive cameras, employing structure-from-motion (SfM) [23] or multi-view stereo (MVS) [30] methods. Unlike active sensors, passive depth estimation is effected by untextured regions and matching errors which can result in many outliers and missing data. Therefore most systems regularize depth maps using smoothness priors, and even optimize over multiple frames using photo-consistency, visibility, and shape priors, before performing surface reconstruction [30, 24, 23, 21].

Recently, due to increased computational capabilities, many new passive systems have pushed further towards live reconstructions. Parallel Tracking and Mapping (PTAM) [13] demonstrated extremely robust sparse tracking, due to real-time performance but only provided sparsely mapped 3D points. Merrell et al. [17] compute noisy depth maps from an image sequence and merge several

*e-mail:vpradeep@microsoft.com

neighboring depth frames to generate fused dense reconstructions. The main objective in the fusion method is to reduce the number of freespace violations. The GPU-accelerated implementation is capable of merging depth maps in real-time. Newcombe et al. [19] propose to incrementally build a dense 3D mesh model from images by utilizing fast and GPU-accelerated techniques for dense correspondence estimation and PTAM-based tracking.

Sparse keypoint-based tracking systems have limitations in textureless scenes, which is addressed by Dense Tracking and Mapping (DTAM) [21]. In DTAM the camera pose is tracked robustly using a dense, whole image alignment method. To generate depth maps, DTAM incrementally builds a cost volume [25] from many data samples, which is continually regularized using a global optimization. This approach takes inspiration from Zach et al. [35], which adds spatial regularization and robustness to the Curless' and Levoy's volumetric method by formulating the depth map fusion problem as TV-L^1-type optimization task, which also can be solved efficiently on the GPU [34]. Whilst producing impressive results, these approaches however carry computational complexity imposed by the global optimization.

In this paper we present a new system called MonoFusion, which allows a user to build dense 3D reconstructions of their environment in real-time, utilizing only a single, off-the-shelf camera as the input sensor. The camera could be one already available in tablet or a phone, or a peripheral web camera. No additional input hardware is required. This removes the need for power intensive active sensors that do not work robustly in natural outdoor lighting. Using the input stream of the camera we first estimate the six degree-of-freedom (6DoF) pose of the camera using a hybrid sparse and dense tracking method. These poses are then used for efficient dense stereo matching between the input frame and a key frame (extracted previously). The resulting dense depth maps are directly fused into a voxel-based implicit model (using a computationally inexpensive method) and surfaces are extracted per frame. The system is able to recover from tracking failures as well as filter out geometrically inconsistent noise from the 3D reconstruction.

Compared to existing approaches, our system avoids expensive global optimization methods for depth computation or fusion such as TV-L^1. Further, it removes the need for a memory and compute intensive cost volume for depth computation. This leads to a simple and efficient system that makes real-time dense 3D reconstructions further accessible to researchers. This paper details the algorithmic components that make up our system and a GPU implementation of our approach. Qualitative results demonstrate high quality reconstructions even visually comparable to active depth sensor-based systems such as KinectFusion.

2 System Overview

MonoFusion works off a live stream of images of a scene from a single, moving camera to generate and maintain a live, dense 3D reconstruction of the scene. The live processing of the image data can be visualized as being split across three computation blocks. This processing scheme is illustrated in Figure 2, and the corresponding blocks are described in detail in the sections that follow. In summary, the live image stream is fed through a hybrid, keyframe based feature tracking and mapping system, that is primarily used to estimate the camera pose trajectory in 6DoF. The camera pose at a given time is used to search for a keyframe (cached as part of the tracking process) with a corresponding camera pose that is optimal (for stereo matching) in terms of baseline and image overlap with the current image frame. This pair of frames and camera poses are used to compute a dense depth map for the current frame by applying a real-time variant of PatchMatch stereo [3]. Per-frame dense depth maps are computed in this manner and ultimately, integrated into a voxel-based representation of the scene surface applying volumetric fusion.

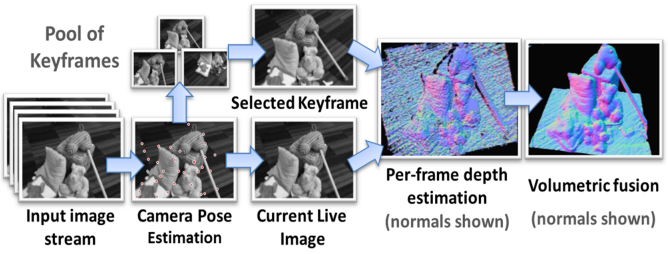

Figure 2: System overview. The 6DoF camera pose trajectory is recovered from a live sequence of images. The online pose information is used to construct per-frame dense disparity maps using PatchMatch stereo and volumetric fusion applied over this data to reconstruct the underlying scene geometry.

3 Camera Pose Tracking

Given a sequence of images $\{I_0, I_1, ..., I_t\}$ over a period of time t, the camera pose tracker estimates the 6DoF camera poses $\{T_0, T_1, ..., T_t\}$, where $T_i, i = 0, 1, ..., t$ represents the 3×4 pose matrix consisting of a 3×3 rotation R and a 3×1 translation vector t. Note that we assume that the camera's intrinsics are already estimated through a checkerboard-based calibration procedure. The process of tracking consists of detecting salient corners every frame of the image stream. By matching them against a persistent map of 3D landmarks (which is also created, updated and maintained by the tracker) the camera pose at the current frame may be estimated. For scale-invariant salient corner detection and matching, an input image I_i is first converted into a 3-level multi-scale Gaussian pyramid [16] and the FAST interest point detector [28] is applied over each level in the pyramid. Further details of the tracking pipeline follow.

Map Initialization. The entire system is initialized by first bootstrapping the camera pose tracker to construct an initial 3D landmark map for tracking. For the first few frames, patches of size $M \times M$ are extracted around the FAST interest points and tracked. During this stage, the user moves the camera slowly across the scene (pure translation is not necessary, but the system will automatically bootstrap only once a sufficient motion baseline is reached). At this stage, since the camera is moving slowly and over a small distance, it can be safely assumed that the patches do not undergo significant appearance change from frame to frame. Hence, a simple Zero Mean Normalized Cross Correlation score (ZNCC) can be utilized in conjunction with inverse match checks to yield a set of matched 2D features between the initial frame and the current frame i. For every frame during this initialization phase, the five point algorithm [22] is applied to the correspondence set in a RANSAC [7] setting to compute the essential matrix $E_{0,i}$ (since the camera is already calibrated) . This essential matrix yields the relative pose transformation between the two viewpoints by computing its singular value decomposition (SVD). This initial pose information is then used to triangulate the matched features in the correspondence set and generate the map $\mathscr{M}$ consisting of $\mathscr{N}$ landmarks, $\{\mathscr{L}_0, \mathscr{L}_1, ..., \mathscr{L}_N - 1\}$, where each landmark $\mathscr{L}_j$ computed at feature j is given by:

$$\mathscr{L}_j = \{X_j, Y_j, Z_j, P_j\} \tag{1}$$

Here, (X_j, Y_j, Z_j) represents the triangulated 3D coordinate of the feature in the coordinate system of the reference frame, and P_j is corresponding $M \times M$ patch centered on the feature pixel from the latest viewpoint image I_i. The patch is stored as a pointer to the appropriate memory block in the Gaussian pyramid of I_i. Before performing the feature triangulation, an arbitrary scale factor

is applied to the translation vector to determine the overall scale of the map. Note that there are alternative ways in which the tracker might be bootstrapped. For instance, PTAM employs a homography based decomposition, but our approach yields robust results using a simpler bootstrapping technique.

Tracking and Pose Estimation. As mentioned at the beginning of this section, multi-scale FAST corners extracted from the current image frame are matched against the 3D landmark map to compute the camera pose. Let us assume that at frame i, we have a guess of the current camera pose T_i^*. Denoting the camera intrinsic matrix as K, we can project any map landmark $\mathscr{L}_j$ into the image plane I_i:

$$I_i(\mathscr{L}_j) = \pi(KT_i^* \begin{bmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{bmatrix}), \tag{2}$$

where $\pi(x) = (x/z, y/z)$ describes the projection of a 3D point x onto the image plane. From the FAST corners extracted in the current image, we extract $M \times M$ patches P_k for those corners that lie within a Euclidean distance threshold of $I_i(\mathscr{L}_j)$ on the 2D image plane. Defining this collection of candidate matches for landmark j as K_j, the best match m_j is defined as the one that minimizes the ZNCC score between the landmark patch P_j and the candidate patch P_k over all candidates in K_j

$$m_j = \underset{k \in K_j}{\arg\min} - \sum_{p \in P_j} \frac{(P_j(p) - \overline{P_j}) \cdot (P_k(p) - \overline{P_k})}{\sigma(P_j) \cdot \sigma(P_k)}. \tag{3}$$

We reject matches that have ZNCC scores above a threshold and perform inverse match checks as well to remove any outliers. Finally, we fit a quadratic function to a window around the best matching pixel and refine the match to sub-pixel accuracy. Given a set of such 3D-2D matching pairs, we employ the three point algorithm in a RANSAC setting, using the Cauchy M-Estimator for error cost computation (instead of mean squared error) to estimate the final pose T_i.

The generation of a good pose estimate T_i^* prior to obtaining the final pose is critical for the agility of the tracker. We compute this guess in two steps. First, similar to PTAM [13], we employ the method of Benhimane and Malis [2] to perform whole image alignment over downsampled images to estimate the camera rotation between the current and previous frame. In the second step, this rotation guess is used to compute T_i^* by finding matches for a few landmarks from the visible set. The landmarks are selected based on how successful their past measurements have been (the ZNCC scores from previous matches are stored) and instead of the five point algorithm at this stage, a minimization procedure that reduces the Euclidean difference between predicted (based on the rotation estimate and pose T_{i-1}) and observed feature locations is used.

Map Updates and Maintenance. From Eq. 2, it is clear that new landmarks have to be continually added to the map as the camera explores more regions in the scene in order to ensure that there always are enough map points visible in the current view. Furthermore, revisiting preexisting landmarks presents a good opportunity to update their 3D locations as well as filter out drift induced errors in the camera trajectory estimate by employing global constraints on the structure of the scene and motion parameters. The map keeps a list of keyframes $\{\mathscr{K}_0, \mathscr{K}_1, ...\}$, where each keyframe $\mathscr{K}_i$ is a collection of the source image I_i, the estimated pose T_i and corresponding landmark set $\mathscr{L}_j, j = 0, 1, ...$ The creation of a keyframe takes place when the amount of landmarks visible in the current frame falls below a specified threshold and landmark generation follows the same procedure as described in the map initialization, with the caveat that a sparse bundle adjustment routine [15] is applied over the new and neighboring keyframes to maintain structure consistency. Additionally, during tracking, we maintain a "hit-count" of the number of landmarks from each keyframe that are matched successfully and once this crosses a set number, the sparse bundle adjuster runs a full structure plus pose optimization over the reobserved keyframes and their neighbors, refining the pose and landmark information attached to each keyframe.

4 Depth Estimation

Our approach for estimating depth is essentially based on stereo matching across the live image and a previously selected key frame. Stereo matching is the task of finding corresponding pixels between two images taken from different but known viewpoints. Stereo algorithms can be categorized into global and local methods. Global approaches (see [29] for an overview) formulate an energy function that is minimized taking all image pixels into account. The optimization techniques used to minimize the energy function are oftentimes too slow for real-time applications. However, approximate global optimization methods [9, 32, 6, 27, 18] based on dynamic programming achieve reasonable frame rates but are restricted to low-resolution images and operate on a strongly quantized depth range (typically at 64 discrete depth values). Local stereo algorithms are generally faster than their global counterparts, because they identify corresponding pixels only based on the correlation of local image patches. Many correlation functions can be implemented as a filter with a computational complexity independent of the filter size. For instance, the sum of absolute differences (SAD) corresponds to a simple box filter [29]. Recent real-time stereo approaches focus on filters that weight each pixel inside the correlation window based on image edges, e.g. based on bilateral filtering [12, 26, 36] or guided image filtering [25, 5]. These approaches show good computational performance if the number of depth values is small. Thus, these approaches do not scale well if a high depth precision is required.

Once the tracking is initialized and running, our system starts generating per-pixel depth frames for every new incoming frame in the video stream, I. Using a method described later in this section, we select an image I' from the tracker's list of keyframes that best matches the current frame (appearance based) and yet provides sufficient baseline to perform depth estimation. Given these two images I and I' (with lens distortion removed), our goal is to search for a depth value for each pixel $i = (u, v)$ in image I that has minimal costs among all possible depth values $\mathscr{D}$:

$$d_i = \underset{d \in \mathscr{D}}{\arg\min} C(i, d), \tag{4}$$

where function C returns the costs for a certain depth hypotheses d at pixel i and is based on the ZNCC over image patches. Let I_p be a square patch in image I centered at pixel p and I'_p the projection of this patch into image I' according to depth d:

$$I'_p(i) = I'(\pi(KT\pi^{-1}(i, d))) \forall i \in I_p. \tag{5}$$

As before, K is the intrinsic matrix of the camera and T describes the relative motion between the two cameras. Function $\pi^{-1}(i, d) = dK^{-1}i$ converts pixel i into 3D scene point x according to depth d. Then C is given, similar to Eq. 3, by

$$C(i, d) = - \sum_{j \in I_p} \frac{(I_p(j) - \overline{I_p}) \cdot (I'_p(j) - \overline{I'_p})}{\sigma(I_p) \cdot \sigma(I'_p)}. \tag{6}$$

$I_p(j)$ returns the intensity value in patch I_p at pixel j. $\overline{I_p}$ and $\sigma(I_p)$ denote the mean and standard deviation in patch I_p, respectively.

4.1 Patch-based Optimization

Evaluating Eq. 4 for all possible depth values is prohibitively expensive especially when dealing with high-resolution images and

if high depth precision is required. We tackle this challenge by employing an optimization scheme similar to the one proposed in PatchMatch stereo [3]. This method has a runtime independent of the number of depth values under consideration. PatchMatch stereo alternates between random depth generation and propagation of depth. However, the runtime performance of the algorithm depends on the correlation window size. Further, PatchMatch stereo is an iterative algorithm requiring several passes over the image.

We iteratively generate a solution by alternating between random depth generation and depth propagation between image pixels. This randomized optimization strategy has the advantage that the computational complexity is independent of the number of possible depth values, i.e. only a small fraction of all possible depth values needs to be tested at each pixel. However, due to the iterative nature of the algorithm a multi-core CPU implementation needs several minutes to process a low-resolution frame [3]. Thus this algorithm is not directly applicable in our real-time application scenario.

In contrast to PatchMatch stereo our 3D reconstruction is not based on a single depth map but is generated by fusing multiple depth maps over time. This means that the quality requirements for the individual depth maps are slightly lower. Thus we propose an approximation of PatchMatch stereo that is capable to generate high-quality depth maps in real-time.

Our two major differences to PatchMatch stereo can be summarized as follows. First, PatchMatch stereo estimates the depth *and* surface normal for each pixel, whereas we focus on computing only depth. Though this has the drawback that slanted surfaces are approximated as piecewise-planar, it reduces the search space from three (depth plus normal) to one dimension (depth). As a consequence, the algorithm converges very quickly and does not need to be iterated. Note that due to the fusion of multiple depth maps piece-wise planar artifacts are reduced almost instantaneously. Second, we use ZNCC on small patches for cost computation (Eq. 6). ZNCC compensates for local gain and offset changes and hence gives us the ability to cope with even large radiometric differences in the input images. (Radiometric changes are likely to occur since our input images are captured at different instances in time.) This is in contrast to PatchMatch stereo where the matching costs are based on the sum of absolute difference of the intensity and gradient inside large patches. The large patches used in [3] are prone to the edge fattening problem and therefore adaptive support weights had to be used to attenuate this effect. By using small patches we not only reduce the computational complexity but also diminish the edge fattening problem and relax the piece-wise planar bias in slanted regions. The drawback of a small patch size is that wrong depth estimates may be obtained in untextured regions. However, wrong depth measurements can be easily discarded and the resulting missing data will be filled over time with depth computed from subsequent frames.

We now continue with a description of our method. The assumption of our approach is that the image comprises of relatively large regions of constant depth (we discuss the size of these regions below). Our algorithm starts by assigning a random depth value to each pixel in the image. Although most pixels will be assigned to a wrong depth value, it is likely that at least one correct guess in each region of constant depth is obtained. Note that regions of constant depth can comprise a large number of pixels and hence the chances for obtaining at least one good (i.e. low cost) guess are quite high. Having obtained one or more depth estimates with low costs we aim to propagate these depth values to spatially neighboring pixels. In the following, we discussed the algorithm in more detail.

Random Initialization. We start by testing for each pixel i a number of K random depth hypotheses $\{\mathscr{D}^* = d_i^1, \ldots, d_i^K\}$. The depth for pixel i is chosen as the one with minimum costs: $d_i := \text{argmin}_{d \in \mathscr{D}^*} C(i,d)$.

To analyse the properties of the random initialization let R be a region of constant depth comprising $|R|$ pixels and let $l = |\mathscr{D}|$ be the number of all possible depth values. Then the likelihood P to correctly assign at least one pixel in R to the correct depth is $P = 1 - (1 - 1/l)^{|R| \cdot K}$. Thus in order to obtain the correct depth for at least one pixel in R with a likelihood of P, the region size has to be $|R| \geq log(1-P)/(K \cdot log(1-1/l))$.

For example, let us assume that we would like to reconstruct depth in a range of up to two meters with a precision of $5mm$. Then the total number of possible depth values $l = 400 = 2000mm/5mm$. If $K = 5$ then with 95% probability at least one correct depth is sampled in a 16×16 region. Note that since our method is based on monocular tracking, scale information is not available to set a threshold in metric units. The minimum and maximum depth values obtained during the bootstrapping phase of the pose tracker are used to determine the threshold in our experiments, and therefore, reconstruction resolution is tied to the arbitrary scale factor applied during this bootstrap phase.

Spatial Propagation. After the random initialization step some pixels in the image will already be assigned to depth values with low matching cost. The idea of the spatial propagation is to pass these depth values to spatially neighboring pixels since these are likely to have the same depth (but might currently be assigned to a different depth). To this end, we traverse the image in row-major order starting from the upper-left corner. For the current pixel $i = (u,v)$ being scanned we look up the depth assigned to the left and upper spatial neighbor $q = (u-1,v)$ and $r = (u,v-1)$. The depth at pixel i is then given by $d_i := \text{argmin}_{d \in \{d_i, d_q, d_r\}} C(i,d)$. After processing every pixel, we reverse the propagation: we start in the lower-right corner and propagate the depth values from the pixels right and lower neighbors. The spatial propagation can be regarded as GPU friendly region-growing process since each pixel along a diagonal can be processed in parallel (see [1]).

4.2 Post-processing

The computed depth map can contain artifacts due to the lack of texture or because of occlusions. Therefore we apply two filters that remove erroneous matches.

Matching Cost Filter. The first filter removes pixels with depth values that generate large costs according to Eq. 6. The costs are typically large for pixels whose patches contain little texture or that are affected by occlusions. We mark pixel i as an outlier if its matching costs according to Eq. 6 are larger than $T_{ZNCC} = 0.5$.

Minimum Region Filter. The second filter removes small isolated regions of similar depth (see e.g. Hirschmüller et al. [9]). We first segment the image by grouping spatially neighboring pixels whose depth values differs by less than T_{depth}. The value of T_{depth} is chosen based on the minimum and maximum depth values in the first depth map. We then mark a pixel i as an outlier if it belongs to a segment that contains less than $T_{minsize} = 50$ pixels.

4.3 Keyframe Selection

From the above discussion, it is clear that the quality of depth map computed for the current frame I is closely tied to how we select the matching image I' from the tracker's cached list of keyframes. For stereo matching to work well, we have to guarantee sufficient image overlap while maintaining a good baseline between the two frames that determines depth accuracy. A naive approach is, given the current frame's position vector and those of all keyframes, set I' to be the one that minimizes the Euclidean distance. This guarantees the largest overlap possible, and since keyframes are generated only when the tracker runs of visible map points, a reasonable baseline can be expected. However, due to camera rotation keyframes can be sampled from viewpoints that are in very close proximity to each other and as a consequence we cannot guarantee a sufficient baseline. To overcome this, we collect a group of keyframes

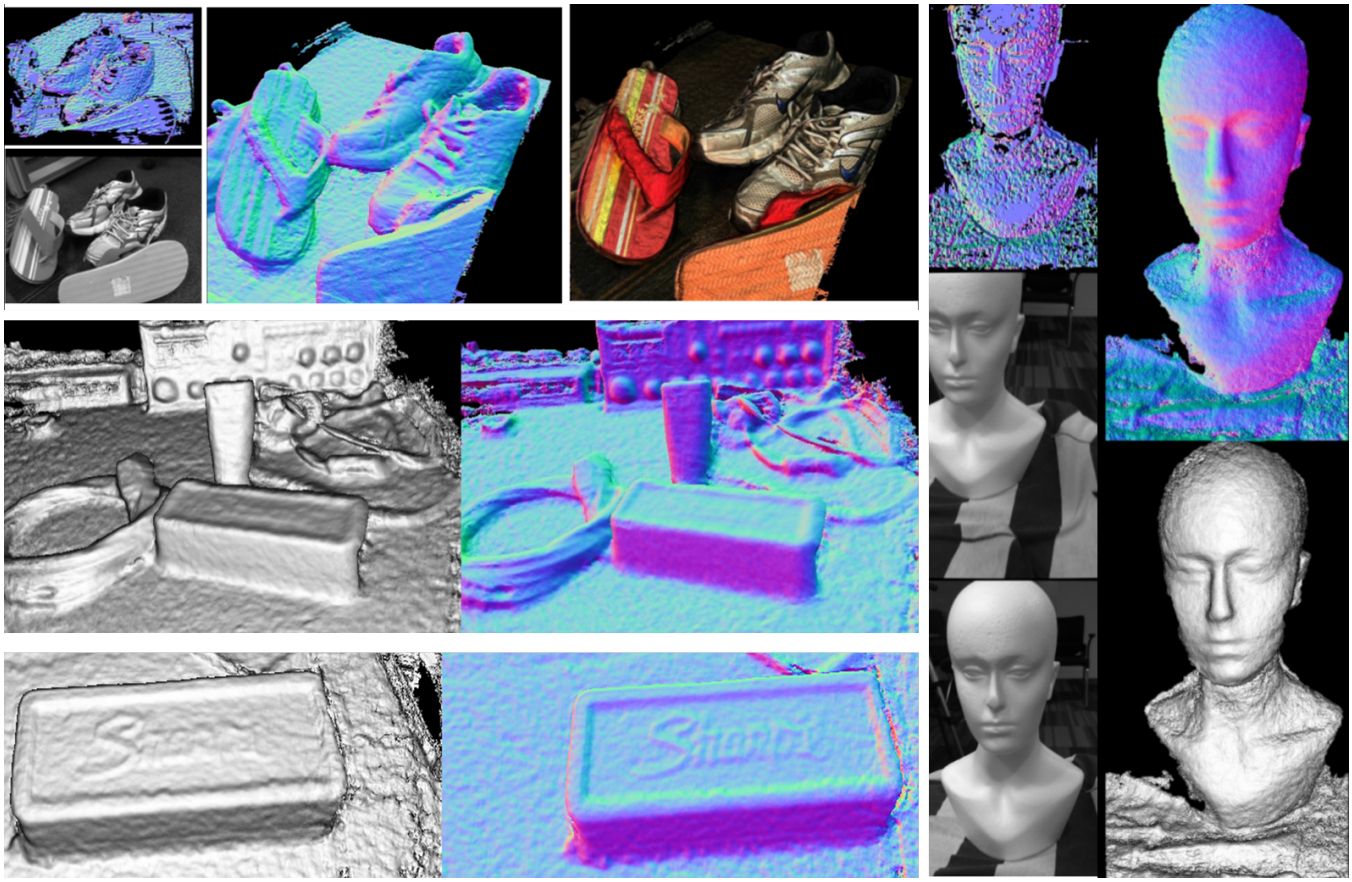

Figure 3: Real-time reconstruction results. Top left: Reconstruction of a set of shoes. Bottom left: A reconstruction of a desktop scene. Note the fine details, such as the embossing on the board rubber. Right: A 3D reconstruction of a mannequin head with minimal texture.

that are within a baseline distance $\mathscr{B}$ of the current frame. Since the tracker provides no metric scale, $\mathscr{B}$ is determined by the camera translation vector and scale factor estimated during bootstrap. Simply selecting the farthest keyframe may result in a too large baseline. Also, keyframes with a large relative rotation can fall in this neighborhood, and pose significant challenges for dense matching. To weigh the candidates, we plot a histogram of the ratio of all landmarks originally detected in each keyframe to those that can be successfully matched in I. Ultimately, the farthest keyframe with a score above a threshold (0.65 in our experiments) is selected as I'.

5 Volumetric Fusion

We adopt the method of Curless and Levoy [4], and encode surfaces implicitly as a signed distance field (SDF). This part of the pipeline is based on the standard KinectFusion system [11, 20] but uses depth maps derived from the moving passive RGB camera. Our system takes these sequence of noisy depth images. We initialize the camera to the origin, which is also the center of the virtual volume's front face. For each frame, we incrementally update the volume by *integrating* (or *fusing*) surface observations into the stored SDF, adding new data into empty regions or denoising existing data. Next, we *raycast* the volume (using the current camera pose estimate), marching individual rays through the grid, to find sign changes in the SDF (the zero-crossing) and extract surface points and normals.

A clear advantage of our local, L^2-based volumetric fusion approach over potentially more robust global optimization techniques (such as TV-L^1 fusion [35] targeting sparser and noiser sets of depth images) is the achieved speed and the ability to process incoming depth maps in an incremental manner. Our approach just maintains a running average of SDF samples during the integration step, and the level of redundancy in our data, the speed of our system, and the quality of our depth maps leads to compelling results despite adopting a far simpler L^2-based SDF denoising approach.

6 Results

In the following examples and the supplementary video we demonstrate compelling reconstructions of a variety of scenes, based on just input from a Microsoft Lifecam. Our implementation runs on a NVidia GTX580 at 30Hz for the full pipeline including tracking and reconstruction. Figures 3 and 1 show how small scenes can be reconstructed in real-time and finally texture mapped using the RGB data. Whilst these scenes require some texture, given that our algorithm allows matching across larger images, we can find correspondences even in parts of the image that appear texture-less at lower resolutions. Another benefit over active triangulation-based sensors such as Kinect is our ability image objects at closer ranges, reconstructing finer details such as the detailing on the shoes and embossing on the board rubber in Figure 3.

As the accompanying video shows these reconstructions can be performed rapidly in real-time. As such this opens up AR scenarios that cannot be directly addressed by active sensors such as ones that require lower power devices or outdoor use. As shown in Figures 1 and 3 the quality of reconstructions is visibly comparable to Kinect-based reconstruction systems such as KinectFusion.

7 Discussion and Conclusions

We have presented MonoFusion a system for markerless tracking and 3D reconstruction in small scenes using just a cheap RGB camera. Compared to existing approaches, our system does not require maintaining a compute and memory intensive cost volume or using expensive TV-L^1 fusion. It tracks and relocalizes the camera pose and allows for high quality 3D models to be captured using commodity RGB sensors.

Whilst our system shows a great deal of potential for widening the applicability of 3D reconstruction it does also raise challenges and areas for future work. Whilst we use off-the-shelf hardware, and could potentially migrate our system to mobile RGB cameras (such as those on tablets and mobiles) we are currently using a GPU which requires a desktop or high end laptop to perform efficiently. One positive aspect of our approach is that it does open up the potential for mobile and tablet cameras to be streamed to a remote server where the computation could occur and the resulting (compressed) data streamed back. This type of scenario would not be possible for active sensors currently as they cannot be readily implemented in mobile devices.

Another limiting factor of any RGB approach is that texture is required for both tracking and depth estimation. This is a limitation over active sensors. However, since our stereo matcher is efficient, one interesting possibility is to experiment with increasing the resolution of our input data to a level where texture begins to be exposed in these problem areas.

Another interesting possibility for future work is to explore the combination of our passive setup with other sensing possibilities either active or passive stereo sensors or even time-of-flight. These systems suffer from a variety of challenges including missing data which could be complemented with our reconstruction approach. Here the process of fusion becomes even more challenging, given the different sensor characteristics.

References

[1] C. Bailer, M. Finckh, and H. P. Lensch. Scale robust multi view stereo. In *ECCV*, 2012.

[2] S. Benhimane and E. Malis. Real-time image-based tracking of planes using efficient second-order minimization. In *IROS*, pages 943–948, 2004.

[3] M. Bleyer, C. Rhemann, and C. Rother. Patchmatch stereo - stereo matching with slanted support windows. In *BMVC*, 2011.

[4] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. Comp. Graph. and Interactive Techn.*, pages 303–312, 1996.

[5] L. De-Maeztu, S. Mattoccia, A. Villanueva, and R. Cabeza. Linear stereo matching. In *ICCV*, 2011.

[6] I. Ernst and H. Hirschmüller. Mutual information based semi-global stereo matching on the gpu. In *ISVC*, 2008.

[7] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. of the ACM*, 24(6):381–395.

[8] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments. In *Proc. Int. Symp. Experimental Robotics*, volume 20, pages 22–25, 2010.

[9] H. Hirschmüller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *CVPR*, 2005.

[10] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[11] 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 *Proc. ACM Symp. User Interface Software and Technology*, pages 559–568, 2011.

[12] M. Ju and H. Kang. Constant time stereo matching. In *MVIP*, 2009.

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

[14] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, et al. The digital michelangelo project: 3D scanning of large statues. In *Proc. Computer Graphics and Interactive Techniques*, pages 131–144, 2000.

[15] M. A. Lourakis and A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.

[16] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, pages 91–110, 2004.

[17] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J. Frahm, R. Yang, D. Nistér, and M. Pollefeys. Real-time visibility-based fusion of depth maps. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[18] M. Michael, J. Salmen, J. Stallkamp, and M. Schlipsing. Real-time stereo vision: Optimizing semi-global matching. In *IEEE Intelligent Vehicles Symposium*, 2013.

[19] R. Newcombe and A. Davison. Live dense reconstruction with a single moving camera. In *Proc. IEEE Conf. Comp. Vision and Pat. Rec.*, pages 1498–1505, 2010.

[20] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. IEEE Int. Symp. Mixed and Augmented Reality*, pages 127–136, 2011.

[21] R. Newcombe, S. Lovegrove, and A. Davison. DTAM: Dense tracking and mapping in real-time. In *Proc. IEEE Int. Conf. Comp. Vision*, pages 2320–2327, 2011.

[22] D. Nister. An efficient solution to the five-point relative pose problem. In *CVPR*, pages 195–202, 2003.

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

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

[25] C. Rhemann, A. Hosni, M. Bleyer, and C. Rother. Fast cost-volume filtering for visual correspondence and beyond. In *CVPR*, 2011.

[26] C. Richardt, D. Orr, I. Davies, A. Criminisi, and N. Dodgson. Real-time spatiotemporal stereo matching using the dualcross-bilateral grid. In *ECCV*, 2010.

[27] I. D. Rosenberg, P. L. Davidson, C. M. Muller, and J. Y. Han. Real-time stereo vision using semi-global matching on programmable graphics hardware. In *SIGGRAPH Sketches*, 2006.

[28] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *ECCV*, 2006.

[29] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 2002.

[30] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. IEEE Conf. Comp. Vision and Pat. Rec.*, volume 1, pages 519–528. IEEE, 2006.

[31] J. Stückler and S. Behnke. Robust real-time registration of RGB-D images using multi-resolution surfel representations. In *Proc. ROBOTIK 2012*, pages 1–4. VDE, 2012.

[32] O. Veksler. Stereo correspondence by dynamic programming on a tree. In *CVPR*, 2005.

[33] T. Weise, T. Wismer, B. Leibe, and L. Van Gool. In-hand scanning with online loop closure. In *Proc. IEEE Int. Conf. Computer Vision Workshops*, pages 1630–1637, 2009.

[34] 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.

[35] C. Zach, T. Pock, and H. Bischof. A globally optimal algorithm for robust tv-l1 range image integration. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[36] K. Zhang, G. Lafruit, R. Lauwereins, and L. Gool. Joint integral histograms and its application in stereo matching. In *ICIP*, 2010.