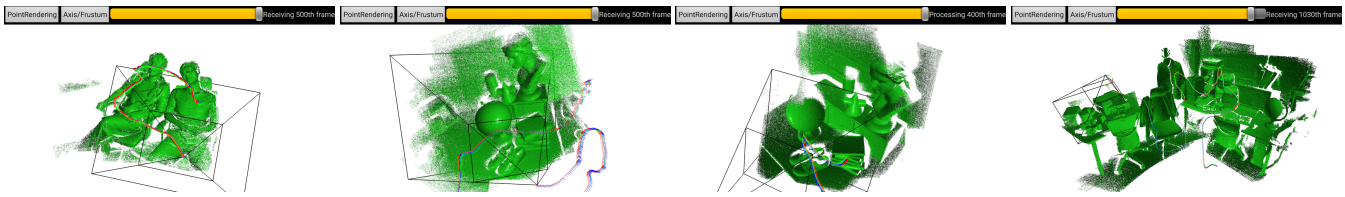
Low-Cost Depth Camera Pose Tracking for Mobile Platforms

Insung Ihm* Youngwook Kim* Jaehyun Lee* Jiman Jeong* Ingu Park†

*Department of Computer Science and Engineering
Sogang University

†NCSOFT Corporation



(a) 2-Men-S (b) Desk-S (c) Desk-K (d) Office-K

Figure 1: Camera tracking results. The presented camera pose estimation technique took roughly 34 ms per input depth frame on a mobile phone when the tracking computations except for the bilateral filtering of raw depth images of 320×240 pixels, which was done in parallel on the GPU, were performed by the mobile CPU alone. Currently in our 3D object scanning system, a user can turn on and off the OpenGL rendering that interactively visualizes progressively growing point-normal sets during camera tracking. Please see also the supplementary materials.

Abstract

The KinectFusion algorithm is now used routinely to reconstruct dense 3D surfaces at real-time frame rates using a commodity depth camera. To achieve robust pose estimation, the method conducts the frame-to-model tracking during camera tracking that must inevitably accompany the memory-bound, GPU-assisted volumetric computations for the model manipulation, to which mobile processors are often more vulnerable than PC-based processors. In this paper, we present an effective camera-tracking method that is based on the computationally lighter frame-to-frame tracking method. This method's tendency toward rapid accumulation of pose estimation errors is suppressed effectively via a predictor–corrector technique. By removing the costly volumetric computations from the pose estimation process, our camera tracking system becomes more efficient in terms of both time and space complexity, offering a compact implementation of depth sensor-based camera tracking on low-end platforms such as mobile devices in addition to high-end PCs.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation—Digitizing and Scanning; I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Tracking H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

1 Problem and Our Contribution

The KinectFusion algorithm [4, 2] is a widely accepted method for depth sensor-based camera tracking, enabling us to track six-degrees-of-freedom camera poses and reconstruct dense surface models at real-time frame rates. Although developed originally for the Microsoft Kinect sensor, it can be adapted effectively for other similar commodity depth cameras in a variety of real-time applications. The two key elements of the KinectFusion algorithm are the frame-to-model correspondence between two sequential frames and the fast iterative closest point (ICP) pose estimation, which together generate an incrementally growing 3D model that is represented implicitly in a volumetric signed distance field.

A linearized computational model that needs a few iterations to solve a 6×6 linear system offers fast computation of an estimate for the camera pose of the current frame. However, because of the linearity of the method, errors tend to accumulate rapidly when the current frame is to be aligned with only the previous frame (*frame-to-frame tracking*), resulting in the estimated camera pose becoming inaccurate within a few frames. This problem was solved by registering the current frame with the 3D model accumulated up to the previous frame (*frame-to-model tracking*), resulting in stable and accurate tracking. While very effective, this approach inevitably demanded the generation of an enhanced depth image for every frame, involving memory-intensive volumetric computations such as signed distance accumulation and ray casting. For real-time applications, this requires accelerated computation via massively parallel GPU hardware. Despite the recent remarkable advances in mobile computing, however, mobile processors are still often vulnerable to such heavy computations in terms of the cost and power consumption.

In this work, we propose a low-cost method for real-time depth-camera tracking that avoids having to maintain the costly volumetric signed distance field while estimating camera poses effectively from live depth images. Our method is based on the linearized ICP model of KinectFusion, but aligns the current frame with previous frames without relying on the surface model built into the volume space. Instead, to suppress the numerical instability problem caused by the deficiencies in the frame-to-frame tracking, we adopt the idea behind *prediction–correction methods*, which has been applied successfully to the numerical integration of ordinary differential equations [3]. By separating the computation of the pose estimation from the volumetric computation for the 3D reconstruction and enhanced depth extraction, our camera tracking method itself becomes computationally more efficient and significantly reduces the memory requirement to maintain the truncated signed distance field, allowing a compact implementation of the depth sensor-based object scanning process on mobile platforms, which are less able to handle high computational complexity and heavy memory access requirements than typical PC platforms.

*e-mail:{ihm, kimyu7, kidsnow, sixzone11}@sogang.ac.kr
†e-mail:ssault@ncsoft.com

2 LINEARIZED ICP MODEL [4]

Consider a live depth stream produced by a moving depth camera, each frame D_k $(k = 0, 1, \cdots)$ of which provides the distance $D_k(\vec{u})$ from the camera plane to the nearest object seen through every pixel $\vec{u}$. When the 3×3 intrinsic matrix of the camera is known, the coordinate $\vec{p}_k(\vec{u})$ of the nearest point, defined in the kth frame's *camera space*, can be obtained via an inverse projection transformation. In addition, a simple divided-difference technique applied to D_k can give an approximation to the normal vector $\vec{n}_k(\vec{u})$ at $\vec{p}_k(\vec{u})$. Let P_k be the set of all point–normal pairs $(\vec{p}_k(\vec{u}), \vec{n}_k(\vec{u}))$ created from D_k.

Given a live depth stream, an essential task is to estimate the *pose* of the depth camera at each time k in a unique *world space*. We express the camera pose as a 3×4 matrix $T_k = \left[R_k \,|\, \vec{t}_k\right]$, where the translation vector $\vec{t}_k$ is the position of the camera center, and the three columns of the 3×3 rotation matrix R_k are the three orthonormal vectors representing the camera's heading. In a frame-by-frame process, the KinectFusion method estimates the camera pose matrix T_k for the current kth frame using the matrix T_{k-1} and the point–normal set P^*_{k-1} obtained from the enhanced depth image that is extracted on the fly from the signed distance field accumulated thus far.

To obtain an estimate for T_k, an initial guess $\tilde{T}^{z=0}_k$ is iteratively refined by estimating a rigid body transformation T^z_{inc} $(z = 1, 2, \cdots)$ such that $\tilde{T}^z_k = T^z_{inc}\tilde{T}^{z-1}_k$ until a satisfactory estimate $\tilde{T}^z_k$ is obtained. In each z-iteration, the point-to-point correspondence between the point–normal sets of the $(k-1)$th and kth frames is established using the fast inverse calibration method [1], where the transformation $\tilde{T}^z_{k\to k-1} = T^{-1}_{k-1}\tilde{T}^{z-1}_k$, composed using the estimated $\tilde{T}^{z-1}_k$ from the previous iteration, is used as the (approximate) rigid body transformation from the kth to the $(k-1)$th camera spaces. Then, the matrix T^z_{inc}, which refines $\tilde{T}^{z-1}_k$ one step further, is calculated by optimizing an energy function in the form of the sum of the squares of the "distance errors" that are approximately expressed as *linear combinations* of the translation displacement (t_x, t_y, t_z) and rotation angles (α, β, γ) of T^z_{inc}. Because of the linearity of the least-squares problem, T^z_{inc} in each z-iteration can be approximated rapidly, simply by solving a linear system of normal equations in six unknowns $(t_x, t_y, t_z, \alpha, \beta, \gamma)$.

3 A MODIFIED FRAME-TO-FRAME TRACKING TECHNIQUE

3.1 Application of a predictor–corrector method

The linearized ICP computation of the KinectFusion method (L-ICP for short) involves an initial guess $\tilde{T}^{z=0}_k$ for T_k being progressively refined as $\tilde{T}^z_k = T^z_{inc}\tilde{T}^{z-1}_k$. In every z-iteration, $\tilde{T}^{z-1}_k$ is used to find the "closest" points between two frames, which are then used to describe the energy function to be optimized. The camera pose T_{k-1} from the previous frame is usually chosen as the initial guess, there being no obvious better alternative. Combined with any possible deficiency in the point–normal set P_{k-1} produced from the single, error-prone depth image D_{k-1}, giving a poor initial value for the linearized computational model often leads to an inaccurate approximation.

The camera poses that are estimated by a simple frame-to-frame tracking scheme may not be sufficiently precise because errors in the resulting matrices tend to accumulate quickly. However, the estimated poses for the next few frames will be fairly close to reality. Therefore, they may be utilized as initial guesses for the z-iterations in later rounds of the L-ICP computation to improve the accuracy of the corresponding camera poses. Note that the L-ICP computation is *implicit* in that the matrix T_k is needed to solve for itself. In principle, the situation is similar to the case of solving ordinary differential equations, where a higher-order solution is often described implicitly. In developing our method, we borrow the idea of a numerical technique called the *predictor–corrector method*, which has been applied successfully to the numerical integration of ordinary differential equations [3].

Figure 2 illustrates the proposed computation pipeline for input (T_{k-1}, P_{k-1}, P_k), where T_{k-1} is a successful previous estimate. With the $(k-1)$th frame as a *reference frame*, the frame-to-frame tracking computation first uses the fast L-ICP method, generating poses $\hat{T}_{k+i}$ for the subsequent frames. This *predictor step* finishes after $m+1$ frames $(m \geq 0)$. The last camera pose $\hat{T}_{k+m}$ is then employed as the initial value in another L-ICP computation to integrate the depth image D_{k+m} of the $(k+m)$th frame against D_{k-1} of the most recent previous reference frame. After the enhanced pose T_{k+m} is obtained in this *corrector step*, the $(k+m)$th frame becomes the reference frame for the subsequent pose estimation. In our modified frame-to-frame tracking approach, only these reference frames, that is, the frames to which the corrector step has been applied, are regarded as legitimate depth frames. It is these frames whose point–normal pairs are used later to reconstruct 3D models.

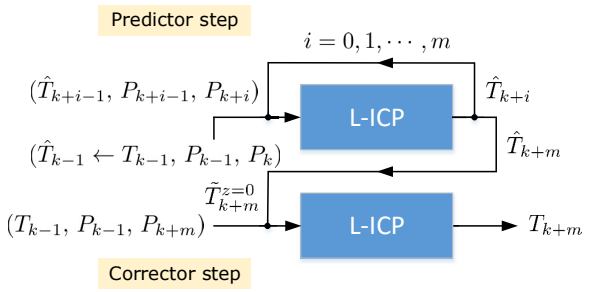


Figure 2: The modified frame-to-frame camera tracking. After the L-ICP computation is repeated for the next $(m+1)$ frames starting from the $(k-1)$th frame (the predictor step), the most recent estimate $\hat{T}_{k+m}$ is used as input to another L-ICP computation between the $(k-1)$th and the $(k+m)$th frames to obtain an error-corrected estimate T_{k+m} (the corrector step).

3.2 Adaptive selection of depth image frames

Taking only the reference frames for further computations has the effect of temporal sampling of the dense depth stream, eventually resulting in efficient processing of the captured data. One way to choose the intervals between the reference frames is to reflect the actual motion of the camera, whereby the predictor step stops if the "predicted" camera pose deviates markedly from that of the last reference frame. Let $T_{k-1} = \left[R_{k-1} \,|\, \vec{t}_{k-1}\right]$ and $\hat{T}_{k+i} = \left[\hat{R}_{k+i} \,|\, \hat{\vec{t}}_{k+i}\right]$ be the pose matrix of the last reference frame and the matrix produced in the predictor step, respectively (see Figure 2). The translation distance and rotation angle between the two poses are then expressed as $||\hat{\vec{t}}_{k+i} - \vec{t}_{k-1}||_2$ and $\cos^{-1}\frac{\mathbf{tr}(\hat{R}_{k+i}R^t_{k-1})-1}{2}$, respectively, where $\mathbf{tr}(\cdot)$ denotes the trace of a square matrix. Therefore, for given distance and angle thresholds ε_d and ε_a, respectively, we exit from the predictor-step iterations if at least one of the following conditions is met: $||\hat{\vec{t}}_{k+i} - \vec{t}_{k-1}||_2 > \varepsilon_d$ and $\mathbf{tr}(\hat{R}_{k+i}R^t_{k-1}) < \varepsilon^*_a$ with $\varepsilon^*_a = 2\cos\varepsilon_a + 1$.

3.3 Normal-based point filtering

The error introduced in the normal vector $\vec{n}_k(\vec{u})$ at $\vec{p}_k(\vec{u})$, estimated by applying a divided-difference scheme to noise-prone back-projected points, often influences significantly the reliability of the ICP computation. In an attempt to supply only reliable point–normal pairs to the L-ICP stage, we modified the normal approximation method as follows. First, two approximations were computed in two different divided-difference directions: $\vec{n}^p_k(\vec{u}) =$

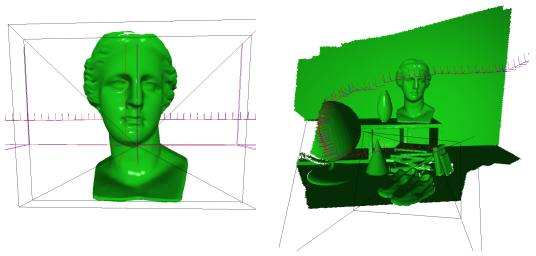
$\frac{\vec{N}_k^p(\vec{u})}{||\vec{N}_k^p(\vec{u})||}$ in the principal axis direction and $\vec{n}_k^d(\vec{u}) = \frac{\vec{N}_k^d(\vec{u})}{||\vec{N}_k^d(\vec{u})||}$ in the diagonal direction, where, for $\vec{u}_{i,j} = (u+i, v+j)$ with $\vec{u} = (u,v)$, $\vec{N}_k^p(\vec{u}) = \{\vec{p}_k(\vec{u}_{1,0}) - \vec{p}_k(\vec{u}_{-1,0})\} \times \{\vec{p}_k(\vec{u}_{0,1}) - \vec{p}_k(\vec{u}_{0,-1})\}$ and $\vec{N}_k^d(\vec{u}) = \{\vec{p}_k(\vec{u}_{1,1}) - \vec{p}_k(\vec{u}_{-1,-1}\} \times \{\vec{p}_k(\vec{u}_{1,-1}) - \vec{p}_k(\vec{u}_{-1,1})\}$. Then, if the angle between $\vec{n}_k^p(\vec{u})$ and $\vec{n}_k^d(\vec{u})$ is greater than a given threshold ε_n, we decide that the captured depth values around the pixel $\vec{u}$ are not sufficiently dependable, and remove its point–normal pair from the ICP computation. Otherwise, their average direction is used as $\vec{n}_k(\vec{u})$ in the subsequent computation. Although this additional normal filtering method itself takes extra computing time, the decrease in the error of the energy function tends to reduce the number of z-iterations and therefore the overall ICP time.

4 Results

We first implemented our method on a PC platform, and the PC version was then ported to a mobile phone that used the Samsung Exynos 8 Octa 8890 chipset equipped with an ARM Mali-T880 GPU. In our current implementation, the entire process was run on the CPU alone except for the bilateral filtering of raw depth images.

4.1 Error analysis using synthetic 3D scenes

To evaluate the effectiveness of the proposed predictor–corrector approach, we performed a quantitative analysis of the translational and rotational errors in the poses produced by variants of the frame-to-frame tracking method. For this experiment, we used two synthetic depth streams with ground truth camera poses, created by rendering the virtual 3D scenes of different complexity using OpenGL and extracting depth values in OpenGL's camera space (see Figure 3).



(a) Virtual Venus (b) Virtual Desk

Figure 3: Two synthetic 3D scenes with camera trajectories.

Figure 4 shows the results on the translational and rotational errors in the poses estimated for the Virtual Desk scene, where both types of errors were usually accumulated rapidly during tracking when the original frame-to-frame tracking method (Frame-to-frame) was applied, with or without normal-based point filtering (NF), resulting in significant artifacts in the reconstructed surfaces. On the other hand, when enhanced with the predictor–corrector method, the frame-to-frame tracking method (Ours) reduced drastically the pose estimation errors, as indicated clearly in the graphs. (Similar results were also obtained for the Virtual Venus scene.)

In all cases, the predictor–corrector mechanism worked as hoped, keeping the errors down to levels much below those for the simple frame-to-frame tracking. In particular, we can observe that the adaptive frame selection method (Adaptive) effectively picked the frames to be corrected based on the estimated errors, and produced more precise and more stable tracking results than the methods (Fixed n) that selected every nth frame as a reference frame. When the adaptive method was applied, 167 and 46 reference frames were selected out of 1000 and 436 total frames for Virtual Venus and the Virtual Desk, respectively, implying that these numbers of additional L-ICP computations were necessary to correct the estimation errors. (The average intervals between them were 5.994 and 9.578, respectively.)

The translational errors of 1.11 mm (ave) and 2.05 mm (max) for Virtual Venus and 0.201 mm (ave) and 0.458 mm (max) for Virtual Desk were quite small, considering that the distances from the camera to the nearest surfaces were confined to a range of 400 mm to 1,400 mm in this experiment. The rotational errors were kept down to 0.0661 ° (ave) and 0.137 ° (max), and 0.0192 ° (ave) and 0.0740 ° (max) for these two scenes, respectively. It is noted that about an order of magnitude smaller estimation errors were achieved for Virtual Desk than for Virtual Venus; this was mainly because more geometric features were represented in the depth images of the former scene, enabling a more stable pose estimation.

4.2 Real scene tests on a mobile phone

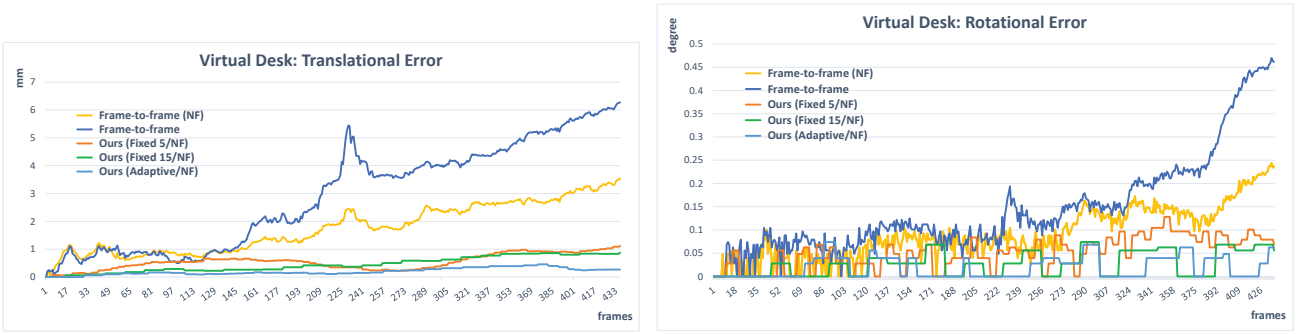
We also tested our camera tracking method on the mentioned mobile phone, where live depth streams at 320×240 resolution were taken at 30 fps in real environments by a mobile device-based Structure sensor and a PC-based Kinect sensor (Kinect for Windows v1). In a real-scene test, applying the normal-based point filtering that rejects back-projected points with inconsistent normal directions is important because our method, relying on the frame-to-frame alignment, is more vulnerable than the frame-to-model alignment to noises in the captured depth images, for which the angle threshold $\varepsilon_n = 10^\circ$ usually worked fine. In addition, the distance and angle thresholds of $\varepsilon_d = 27$ mm and $\varepsilon_a = 3^\circ$ were applied for the adaptive selection of reference frames in the presented results.

Table 1 summarizes the runtime performance measured on the mobile phone for the four depth streams: 2-Men-S and Desk-S from the Structure sensor and Desk-K and Office-K from the Kinect sensor (see Figure 1 for the respective reconstructed point clouds). First, the extra overhead of correcting the pose estimation errors produced by the frame-to-frame tracking is shown in the table (a). Here, "# of L-ICPs" shows the total number of times in which the frame-to-frame L-ICP computation is carried out. Thus, the difference between the frame-to-frame tracking ("original") and our method ("ours") reveals how many reference frames were chosen during the camera tracking. On the other hand, "# of z-iterations" reveals how many times the linear system of normal equations had to be solved, which is a good complexity measure for the entire tracking computation. We find that our solution demanded roughly 1.10 to 1.21 times more tracking computations, respectively, than the faulty frame-to-frame tracking, which is quite acceptable considering the increased precision in the pose estimation.

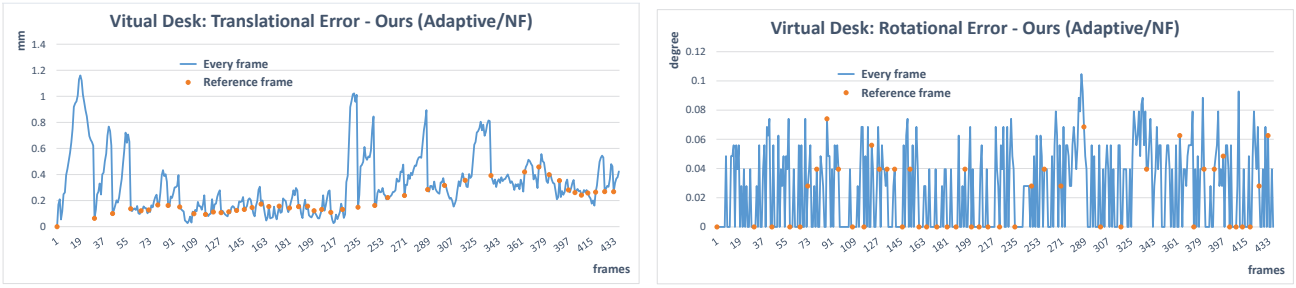
Second, the table (b) presents the actual execution times taken by our method to handle the input depth streams, where the camera tracking computation consumed 30.72 to 37.86 milliseconds per input depth frame on average on the mobile CPU. It is noted that the per-frame tracking time is basically affected by the resolution of input depth images. Thus, when the same resolution was adopted, the camera tracking times for the depth streams from the Kinect sensor were almost the same as those from the Structure sensor, although the former, which is a PC-based sensor, usually captured more refined depth images. It is clear from the achieved timings by the single-threaded mobile CPU version that the modified frame-to-frame approach allowed to effectively perform the camera tracking on the tested mobile phone without concern for the memory-bound GPU computation, for which the mobile device is still more vulnerable than typical PC platforms. We expect that a multithreaded CPU implementation or a GPU optimization will further improve the temporal efficiency on the mobile device.

5 Concluding Remarks

By separating the computation of the pose estimation from that of the 3D reconstruction and enhanced depth extraction in the KinectFusion algorithm, the camera tracking method itself became



(a) Camera pose estimation errors: translational errors (left) and rotational errors (right)



(b) Adaptive selection of reference frames: translational errors (left) and rotational errors (right)

Figure 4: Camera pose estimation errors (Virtual Desk). Frame-to-frame (NF) and Frame-to-frame for the original methods with and without the normal-based point filtering (NF) applied, and Ours (Fixed n/NF) and Ours (Adaptive/NF) for the modified methods with per-every-nth-frame and adaptive corrections.

(a) Overheads of the predictor-corrector method

	# of L-ICPs		# of z-iterations	
	original	ours	original	ours
2-Men-S	499	539	2,465	2,785
Desk-S	499	568	2,463	2,935
Desk-K	400	435	1,901	2,090
Office-K	1,029	1,215	5,360	6,490

(b) Average run times and the sizes of produced point-normal sets

	2-Men-S	Desk-S	Desk-K	Office-K
# of input frames	500	500	401	1,030
# of ref. frames	40	69	35	186
# of pt.s per ref. fr.	28,190.5	34,859.9	35,978.8	27,642.7
# of final points	1,128K	2,405K	1,259K	5,142K
tracking time (ms)	33.02	34.92	30.72	37.86

Table 1: Runtime performance on the mobile phone. In (a), the overheads caused by our tracking method ("ours") that selects the reference frames adaptively are analyzed with respect to the original frame-to-frame tracking ("original"). In (b), the average per-frame timings taken by our method to track the camera is given in "tracking time." "# of pt.s per ref. fr." indicates the average number of point-normal pairs generated per every reference frame, while "# of final points" is for the final sizes of the accumulated point–normal sets.

computationally lighter and significantly reduced the memory requirement, allowing a flexible, computationally lightweight camera tracking software on mobile devices. We have demonstrated that the numerical instability problem of the error-prone frame-to-frame depth-camera tracking method can be eased significantly by correcting the predicted camera poses of adaptively selected depth frames. Furthermore, our approach automatically filtered dense live depth streams, preventing the resulting point–normal cloud from growing excessively. Currently, our 3D object scanning system naturally generates a point-normal cloud as a result of 3D reconstruction, which is dense enough to be able to closely examine the progressively growing surfaces. On the other hand, the possibility is recently shown in [5] that the point-normal pairs, converted into volume space, can be ray-traced interactively on a mobile GPU. If necessary, we could adopt such a surface rendering technique, say, every few other frames, which runs on the GPU in parallel with our CPU-based camera tracking process.

Acknowledgements

This work was partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. NRF-2015R1A2A2A01006590).

References

[1] G. Blais and M. D. Levine. Registering multiview range data to create 3D computer objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):820–824, 1995.

[2] 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 Technology*, pages 559–568, 2011.

[3] D. Kincaid and W. Cheney. *Numerical Analysis*. Brooks/Cole, 1991.

[4] 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 *Proc. 10th IEEE Int. Symp. Mixed Augmented Reality*, pages 127–136, 2011.

[5] P. Ondrúška, P. Kohli, and S. Izadi. Mobilefusion: real-time volumetric surface reconstruction and dense tracking on mobile phones. *IEEE Trans. Vis. Comput. Graph.*, 21(11):1251–1258, 2015.