Real-time Full-view 3D Human Reconstruction using Multiple RGB-D Cameras

Bumsik Yoon[1], Kunwoo Choi[2], Moonsu Ra[2], and Whoi-Yul Kim[2]

[1] Department of Visual Display, Samsung Electronics / Suwon, South Korea bsyoon@samsung.com
[2] Department of Electronics and Computer Engineering, Hanyang University / Seoul, South Korea
{kwchoi, msna}@vision.hanyang.ac.kr, wykim@hanyang.ac.kr

***** Corresponding Author: Whoi-Yul Kim

Received July 15, 2015; Revised August 5, 2015; Accepted August 24, 2015; Published August 31, 2015

** Short Paper*

** Extended from a Conference: Preliminary results of this paper were presented at the ITC-CSCC, Summer 2015. The present paper has been accepted by the editorial board through the regular reviewing process that confirms the original contribution.*

Abstract**:** This manuscript presents a real-time solution for 3D human body reconstruction with multiple RGB-D cameras. The proposed system uses four consumer RGB/Depth (RGB-D) cameras, each located at approximately 90° from the next camera around a freely moving human body. A single mesh is constructed from the captured point clouds by iteratively removing the estimated overlapping regions from the boundary. A cell-based mesh construction algorithm is developed, recovering the 3D shape from various conditions, considering the direction of the camera and the mesh boundary. The proposed algorithm also allows problematic holes and/or occluded regions to be recovered from another view. Finally, calibrated RGB data is merged with the constructed mesh so it can be viewed from an arbitrary direction. The proposed algorithm is implemented with general-purpose computation on graphics processing unit (GPGPU) for real-time processing owing to its suitability for parallel processing.

Keywords**:** 3D reconstruction, RGB-D, Mesh construction, Virtual mirror

1. Introduction

Current advances in technology require three-dimensional (3D) information in many daily life applications, including multimedia, games, shopping, augmented-reality, and many other areas. These applications analyze 3D information and provide a more realistic experience for users. Rapid growth of the 3D printer market also directs aspects of practical use for 3D reconstruction data. However, 3D reconstruction is still a challenging task.

The 3D reconstruction algorithms for given point clouds can be classified according to spatial subdivision [1]: surface-oriented algorithms [2, 3], which do not distinguish between open and closed surfaces; and volume-oriented algorithms [4, 5], which work in particular with closed surfaces and are generally based on Delaunay tetrahedralization of the given set of sample points. Surface-oriented methods have advantages, such as the ability to reuse the untouched depth map and to rapidly reconstruct the fused mesh.

In this paper, 3D reconstruction is proposed by fusing multiple 2.5D data, captured by multiple RGB/Depth (RGB-D) cameras, specifically with the Microsoft Kinect [6] device. The use of multiple capturing devices for various applications means they can concurrently acquire the image from various points of view. Examples of these applications are motion capture systems, virtual mirrors, and 3D telecommunications.

The approach proposed in this manuscript constructs a mesh by removing the overlapping surfaces from the boundaries. A similar approach was proposed by Alexiadis et al. [7]. Meshes generated from the multiple RGB-D cameras can introduce various noise problems, including depth fluctuations during measurement, and holes caused by the interference of infrared (IR) projections from the multiple cameras. The proposed algorithm reduces these issues, by considering the direction of the camera pose and by analyzing various conditions of the captured point clouds.

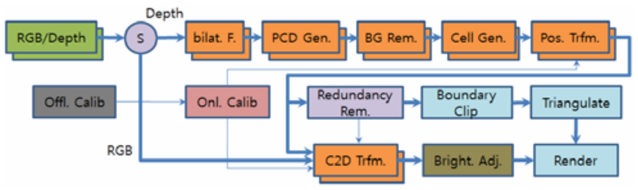


Fig. 1. 3D reconstruction block diagram.

The paper is organized as follows. Section 2 explains the proposed algorithm for 3D reconstruction. Section 3 presents the implementation method of the algorithm. Section 4 discusses the results of the experiment. Finally, Section 5 concludes this manuscript.

2. 3D Reconstruction Algorithm

In the proposed scheme, RGB-D cameras are installed at 90° angles from the adjacent cameras and at a distance of 1 to 2m from the target. The camera parameters and their initial positions are estimated beforehand. If any subtle adjustment to the camera positions is required, optional online calibration can be performed.

At the beginning, depth and RGB data from each camera are captured concurrently in each thread. The captured data are synchronized for subsequent processing. The depth data go through a bilateral filter [8] and are transformed into the point clouds using the calibrated intrinsic depth camera parameters. Then, the point clouds are used to generate cell-based meshes, following the removal of background points.

After the cell generation, each point cloud is transformed to global coordinates with calibrated extrinsic parameters. The redundancies between the point clouds are removed after the transformation by the iterative boundary removal algorithm, and the resultant meshes are clipped together.

The RGB data is transformed to depth coordinates, and the brightness level is adjusted by investigating the intensity of the overlapped regions. Finally, the calibrated RGB data are rendered with the triangulated mesh.

Fig. 1 shows a block diagram of the overall system. Every color of the module represents a CPU thread, and the bold and thin lines indicated in the figure show the flow of data and parameters, respectively.

2.1 Calibrations

A set of checkerboard images is captured from RGB/Depth cameras to estimate the intrinsic and extrinsic parameters, for each camera, using a Matlab camera calibration toolbox. For the depth camera calibration, IR images are used instead of the depth images, because the corner points of the checkerboard cannot be detected in a depth image.

In addition to the depth camera parameters, the shifting error between the IR and depth [9] is considered, because the mapped color usually does not match the point cloud, as shown in Fig. 2(c). Vertices of a colored cube

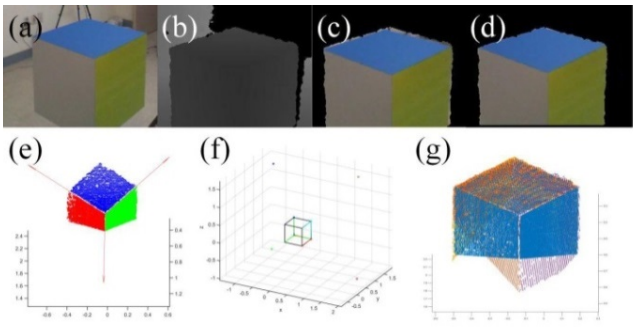


Fig. 2. Calibration process (a) RGB image, (b) Depth image, (c, d) RGB-D mapped image before and after IR-depth shift correction, (e) Edge vectors from the point cloud of a cube, (f) Multi-Kinect registration result, (g) Point cloud aggregation.

(50×50×50cm) from the IR and depth images are found to estimate the shifting value. The intersection point of the three edges in the IR image corresponds to the vertex of the cube in the depth image. The vertex can be found via the intersection of the estimated three planes. The found offset is applied in the color-to-depth transformation module.

Usually, the extrinsic parameters between two cameras can be estimated by finding the corresponding corners of the checkerboard images from the cameras. However, if the angle between the two cameras is too large, this method is difficult to use due to the narrow common view angle. Therefore, a multi-Kinect registration method is proposed that uses a cube for the calibration object. It needs only one pair of RGB/depth images per RGB-D camera in one scene.

Fig. 2(e) shows the edge vectors and the vertex, identified by the colors of the intersecting three planes for one camera. The found edge vectors are transformed to the coordinates of a virtual cube, which has the same size as the real cube so as to minimize the mean square error of the distances for four vertices viewable from each camera. The registered cube and the estimated pose of the depth cameras are shown in Fig. 2(f), and the aggregated point cloud is given in Fig. 2(g).

Online calibration for the extrinsic parameters can be performed if a slight change in the camera positions occurs by some accidental movement. An iterative closest point (ICP) [10] algorithm could be applied for this purpose. However, there are two kinds of difficulty with traditional ICP aligning all the point clouds in the proposed system. First, traditional ICP works only in a pairwise manner. Second, the point clouds do not have sufficient overlapping regions to estimate the alignment parameters.

To resolve these problems, a combined solution of generalized Procrustes analysis (GPA) [11] and sparse ICP (S-ICP) [12] is adopted. The basic concept is as follows.

1. Extract the common centroid set that would become the target of S-ICP for all the point clouds.
2. Apply S-ICP on the centroid set for each point cloud.

The difference between GPA presented by Toldo et al. [11] and our proposed method is that only left and right point clouds are used for centroid extraction, as seen in Fig.

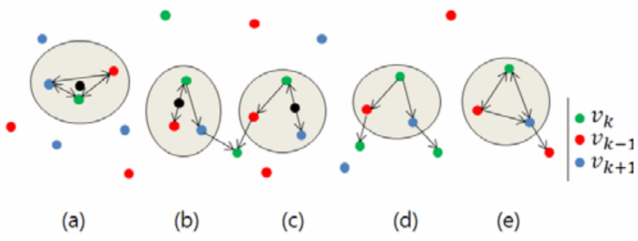


Fig. 3. Mutual neighboring closest points (a, b, c) Valid cases, (d, e) Invalid cases.

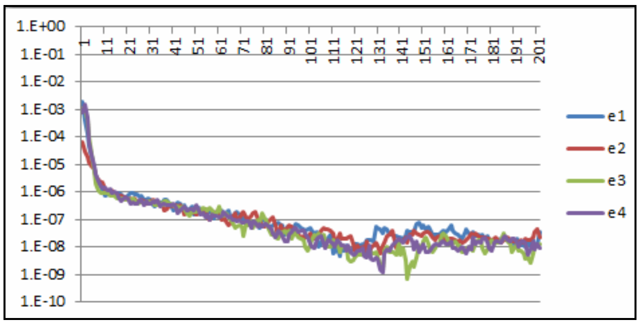


Fig. 4. Registration errors.

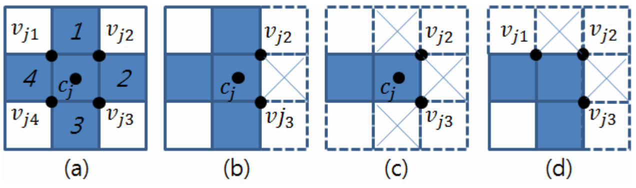


Fig. 5. Various cell types (a) No boundary cells, (b, c, d) Examples of directional boundary cells.

3. The direction of the arrow indicates its closest vertex in the neighboring point cloud, and the black dot indicates its centroid.

S-ICP is repeatedly performed until all of the maximum square residual errors of the pairwise registration become less than a sufficiently low value. Fig. 4 shows the transition of the errors when three of four cameras are initially misaligned by about 10cm and at 5° to the arbitrary direction.

2.2 Cell Based Mesh

A cell-based mesh (CBM) is used for redundancy removal, rather than the unordered triangle-based mesh, because CBM is quick to generate, and it is also feasible to utilize the grid property of the depth map. The projected location of the cell and its boundary condition can be examined rapidly, and this is used frequently in the proposed algorithms.

A cell is constructed if all four edges surrounding the rectangular area in the depth map grid are within the Euclidean distance threshold D_m. During the boundary removal stage, the center of the cell is used, which is calculated by averaging the positions of the four neighboring vertices around the cell (Fig. 5(a)). The normal of each cell is also generated by calculating the cross product of two vectors of the three vertices around the cell.

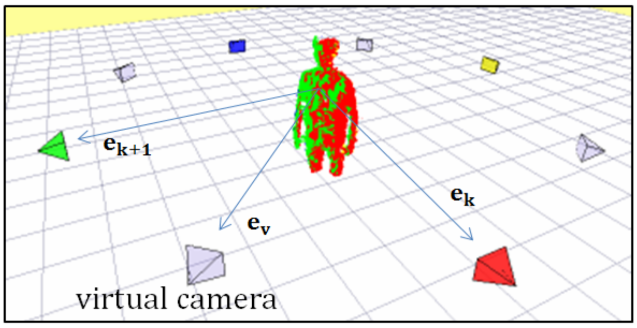


Fig. 6. Camera positions.

The boundary cell is simply defined if the cell does not have any surrounding cells sharing an edge. The direction of the boundary cell is defined as the outward direction from the center to the boundary. For horizontal/vertical boundary cells, the direction is calculated as the weighted sum of vectors from the center to the vertices of the boundary edge (Figs. 5(b) and (c)):

$$\mathbf{b_j} = \left|v_{j2} - c_j\right|\left(v_{j3} - c_j\right) - \left|v_{j3} - c_j\right|\left(v_{j2} - c_j\right). \tag{1}$$

For the diagonal boundary cell, the direction is calculated as the weighted sum of vectors from the center to the diagonal vertices (Fig. 5(d)):

$$\mathbf{b_j} = \left|v_{j2} - v_{j3}\right|\left(v_{j1} - v_{j2}\right) - \left|v_{j1} - v_{j2}\right|\left(v_{j2} - v_{j3}\right). \tag{2}$$

There are undecidable one-way directional boundary cells, such as a thin line or a point cell. These cells are categorized as non-directional boundary cells and are dealt with accordingly.

2.3 Redundant Boundary Cell Removal

The transformed cells may have redundant surfaces that overlap surfaces from other camera views. The redundant cells are removed by the redundant boundary cell removal (RBCR) algorithm. RBCR utilizes the direction of the virtual camera $\mathbf{e}_v$ (Fig. 6), which is the middle view of its neighboring camera. Using this direction, we can effectively estimate the redundant surfaces, minimizing the clipping area. It is also used as the projection axis for 2D Delaunay triangulation.

Let $\mathcal{M}_k$ be the cell mesh generated by camera k, let $C_{k,j}$ be the jth cell in $\mathcal{M}_k$, and let $c_{k,j}$ be the center of cell $C_{k,j}$. The index k is labeled in the order of circular direction. Assuming that $C_{k,j}$ in $\mathcal{M}_k$ is a boundary cell, it is deemed redundant if a corresponding $C_{k+1,m}$ can be found that minimizes the projective distance, d_p, with the constraint that the Euclidean distance (d_a) between the center of the cells should be smaller than the maximum distance, D_a:

$$C^*_{k+1,m} = \underset{C\in\{C|d_a<D_a, C\in\setminus M_{k+1}\}}{\operatorname{argmin}} \left(d_p\right) \qquad (3)$$

The projective distance d_p is defined as follows:

$$d_p = d_a - |(c_{k+1,m} - c_{k,j})\cdot \mathbf{e}_v| \qquad (4)$$

where $\mathbf{e}_v$ is found by spherical linear interpolation, or "slerp" [13], with angle Ω between camera direction $\mathbf{e}_k$ and $\mathbf{e}_{k+1}$:

$$\mathbf{e}_v = \frac{\sin(\Omega/2)}{\sin(\Omega)}\left(\mathbf{e}_k + \mathbf{e}_{k+1}\right). \qquad (5)$$

To find C^*, a projection search method is adopted, i.e., $c_{k,j}$ is projected to the target view of $\mathcal{M}_{k+1}$, and the cells of $\mathcal{M}_{k+1}$, in the fixed-size window centered on the projected $c_{k,j}$, are tested for the conditions.

Once C^* is found, the corresponding $C_{k,j}$ is considered a potentially redundant cell. The additional conditions are tested to decide if the cell is truly redundant, and hence removable.

If the found C^* is not a boundary cell and the normal is in the same direction, it is redundant because $C_{k,j}$ is on or under the surface, not the cell of a thin object. Or, if C^* is a directional boundary cell, $C_{k,j}$ is redundant when $C_{k,j}$ is close enough to C^* so that d_p is smaller than the maximum projective distance (D_p), and the boundary directions are not in the same direction. This could be regarded as the depth test in ray-tracing for the direction of $\mathbf{e}_v$ of the boundary cell.

The way mutual directionality is decided is by the sign of the inner product for the two directions.

In one loop, RBCR is performed through all ks, for the outermost boundary cells in $\mathcal{M}_k$ w.r.t. $\mathcal{M}_{k+1}$, and vice versa, and is applied iteratively until no cells are removed.

2.4 Boundary Clipping

In this stage, any boundary cell in $\mathcal{M}_k$ within distance D_a from the boundary of $\mathcal{M}_{k+1}$ is collected with the same search method of RBCR.

The collected cells are disintegrated to the vertices, and are orthogonally projected to the plane of the virtual camera. Then, the projected points are triangulated via 2D Delaunay algorithm.

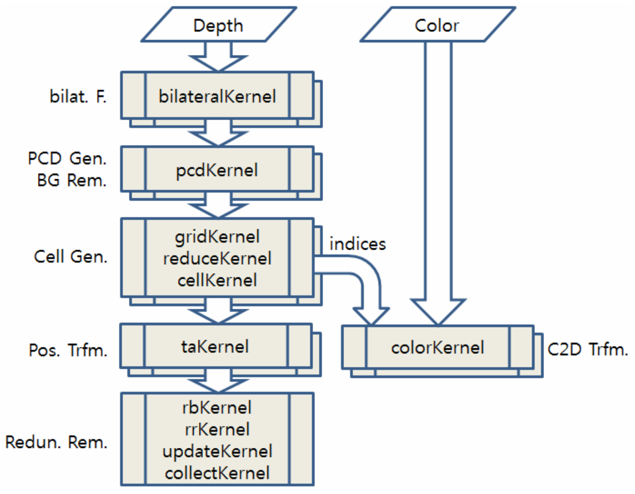


Fig. 7. CUDA kernel composition.

2.5 Triangulation

Except for the triangulated cells in the previous boundary clipping stage, all the other cells are simply triangulated by dividing the cell into the two triangles. The shorter diagonal edge is selected for triangulation.

2.6 Brightness Adjustment

Although the Kinect device provides an auto-exposure functionality, it is not sufficient to tune the multiple RGB cameras. The brightness is tuned online by multiplying the correction factor. Each factor is calculated by comparing the intensity of the overlapped region with the mean intensity of all overlapped regions. The overlapped regions can be directly extracted from the RBCR operation.

The propagation error from all the cameras is distributed to each correction factor so that the overall gain is 1.

3. Implementation

Among the modules of Fig. 1, the bilateral filter through the position transform, the redundancy removal, and color-to-depth transform modules are implemented under the Compute Unified Device Architecture (CUDA) [14]. The rendering module is implemented with OpenGL and all other modules with the CPU.

Fig. 7 shows all of the implemented CUDA kernels that correspond to the logical modules in Fig. 1.

bilateralKernel is configured to filter one depth with one thread each. The radius and the standard deviation of the spatial Gaussian kernel were set to 5 pixels and 4.5 pixels, respectively. The standard deviation of the intensity (i.e. depth value) Gaussian kernel was set to 60mm.

pcdKernel generates point cloud back-projecting of the depth pixels with the calibrated intrinsic parameters. The kernel also eliminates the background depth pixels with a frustum of near 0.5m and far 2.5m.

The cell generation module consists of three kernels.

gridKernel marks the valid vertices within distance D_m. As the neighboring relationship is needed to check the validity, four vertices are marked with an *atomicOr* barrier function if they turn out to be valid. *reduceKernel* reduces the grid vertices to a reduced stream, generating the indices for the marked vertices. *cellKernel* constructs the cell information if all of the neighboring vertices are valid. The constructed cell information includes both the normal and the center of the cell.

The positional transformation is done in *taKernel*. It includes vertex, normal, center transform and the cell projection. Although the kernel could be implemented with a general transform kernel, as the transforms use the same parameter, it is more efficient to process them all at once by reducing the kernel launch time, rather than by calling the general purpose kernel multiple times.

The RBCR algorithm is designed to run concurrently for the four pairs of the mesh by using the CUDA stream feature, not using the CPU thread, because the status of the cells needs to be synchronized for every loop. *rbKernel* just removes the first outermost boundary cells because the measured boundary depth tends to be considerably inaccurate.

The RBCR loop runs with the two CUDA kernels.

- *rrKernel*: Searches (3) and marks the flag for the cells to be removed.
- *updateKernel*: Removes all the marked cells and returns the number of removed cells.

The two-kernel approach makes the mesh maintain the consistency of the boundary condition in a loop. The search function in *rrKernel* is designed to use 32 concurrent threads per cell for a 16×16 search window. It leads to loop 8 times for one complete search, and to use 32 elements of shared memory for intermediate storage of the partial reduction. The grid size is defined as the number of cells (N_c) divided by the number of cells per block (N_{cpb}). N_{cpb} is tuned to 16 as a result of performance tuning that maximizes the speed.

The synchronization of the cell status is done automatically when the remove counter is copied from the device to the host with the default stream.

To accelerate RBCR and keep a constant speed, the loop is terminated after the eighth iteration (*max_loop*=8) and one more search is done for all the remaining cells, including the non-boundary cells.

The boundary cells of the RBCR results are collected with *collectKernel* by a method similar to *rrKernel* but without the iterative loop.

colorKernel maps the color coordinates to the depth coordinates followed by correction of the radial, tangential distortions, and IR-depth shifting error using the calibrated parameters. The operation is performed only for the reduced cells.

The boundary clipping module runs on CPU threads other than the RBCR thread to reduce the waiting time for RBCR. The Delaunay triangulation (DT) algorithm is implemented with the Computational Geometry Algorithms Library (CGAL) [15]. As DT generates the convex set of triangles, long-edged triangles ($> D_t$) are eliminated after the triangulation.

We adapt the sparse ICP module [16] using an external kd-tree for mutual neighboring of closest points. The point-to-point $\ell_{0.4}$-ICP is used for optimization, with max inner and outer iterations of 1 and 20, respectively.

The parameter values used in this paper are given in Table 1.

The resolutions for input depth and RGB are both 640×480, and the equipment used for the implementation was a desktop PC with an Intel i7 3.6GHz core and an NVidia GTX970 graphics card.

Table 1. Experiment Parameters.

Parameters	Values
D_m (max cell edge)	2cm
D_a (max Euclidean distance)	3cm
D_p (max projected distance)	0.5cm
D_t (max triangle edge length)	3cm

Table 2. Latencies.

Modules	Latencies
Sync	16.2ms
Cell Gen.	11.2ms
Pos. Trfm.	8.1ms
Redun. Rem.	26.5ms
Triangulation	24.3ms
Total	86.3ms

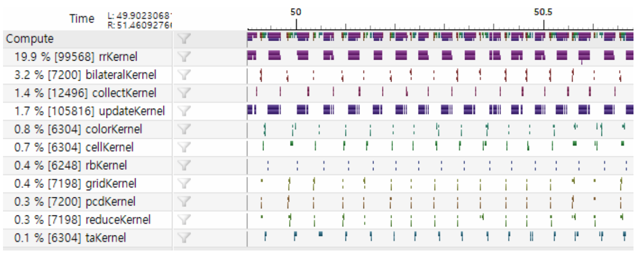


Fig. 8. Performance analysis.

4. Results

Fig. 8 gives the performance analysis results of NVidia Nsight for the implemented kernels. As expected, it shows that *rrKernel* is computationally the most expensive kernel, as expected. The timeline shows that the speed of the overall system is approximately 21fps. The latencies measured at the end of each module are described in Table 2.

Fig. 9(a) shows various views of the reconstructed human mesh that can be seen on the run. The bottom row is the color map of the reconstructed mesh, where color represents the mesh from the corresponding camera. The thin purple line indicates the clipped area. Compared to the original unprocessed mesh in Fig. 9(c), we can see that the

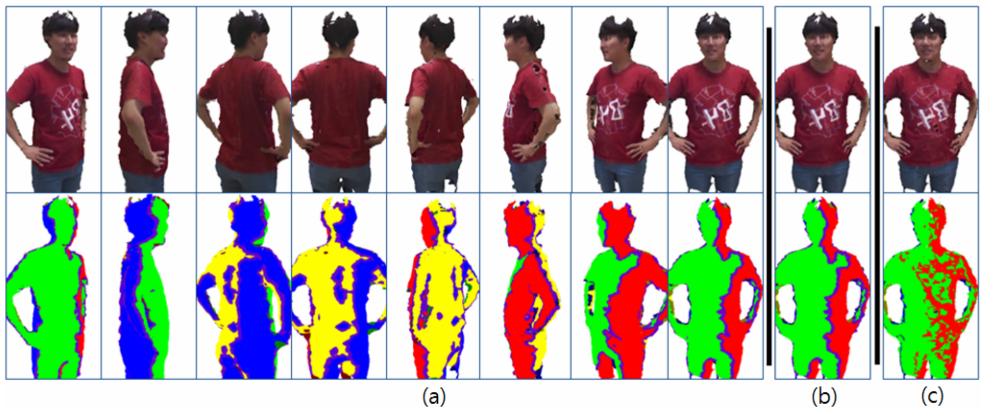


Fig. 9. Result of various views (a) *max_loop* = 8, (b) *max_loop* = 24, (c) The original.

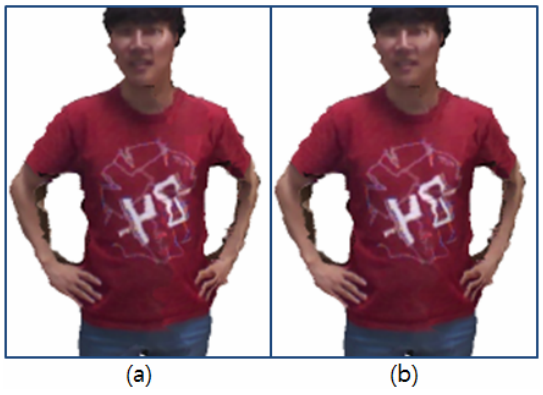


Fig. 10. Result of brightness adjustment (a) before, (b) after.

resultant mesh has no redundancies and is clipped cleanly. Fig. 9(b) is the result of RBCR when *max_loop* is equal to 24, showing almost no difference when *max_loop* is equal to 8.

Fig. 10 gives the result of the brightness adjustment showing that the mismatched color in the cloth is effectively corrected.

5. Conclusion

In this paper, it is shown that the proposed algorithm and the implementation method could reconstruct a 3D mesh effectively, supporting a 360-degree viewing direction with multiple consumer RGB-D cameras. The proposed calibration method, which uses a cube as a calibration object, could estimate the color/depth camera parameters and the global position of the cameras effectively, accommodated by the online calibration method that exploits mutual neighboring closest points, and a sparse ICP algorithm. The constructed mesh had no redundancies after application of the proposed algorithm, which iteratively removes the estimated redundant regions from the boundary of the mesh. In addition, the proposed 3D reconstruction system could adjust the mismatched brightness between the RGB-D cameras by using the collateral overlapping region of the redundancy removal algorithm. The overall speed for implementation was 21fps with a latency of 86.3ms, which is sufficient for real-time processing.

References

[1] M. Botsch, et al., *Polygon Mesh Processing*, AK Peters, London, 2010. Article (CrossRef Link)

[2] H. Hoppe, et al., “Surface reconstruction from unorganized points,” SIGGRAPH ’92, 1992. Article (CrossRef Link)

[3] R. Mencl and H. Müller. Graph–based surface reconstruction using structures in scattered point sets. In Proceedings of CGI ’98 (Computer Graphics International), pp. 298–311, June, 1998. Article (CrossRef Link)

[4] B. Curless and M. Levoy, “A volumetric method for building complex models from range images,” SIGGRAPH ’96, 1996. Article (CrossRef Link)

[5] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson Surface Reconstruction,” *Proc. Symp. Geometry Processing*, 2006. Article (CrossRef Link)

[6] Microsoft Kinect Article (CrossRef Link)

[7] D. Alexiadis, D. Zarpalas, and P. Daras, “Real-time, full 3-D reconstruction of moving foreground objects from mul-tiple consumer depth cameras,” *IEEE Trans on Multimedia*, vol. 15, pp. 339−358, Feb. 2013. Article (CrossRef Link)

[8] C. Tomasi and R. Manduchi, “Bilateral filtering for

gray and color images," in *Proc. of the ICCV*, 1998. Article (CrossRef Link)

[9] J. Smisek, M. Jancosek, and T. Pajdla, "3D with Kinect," 2011 ICCV Workshops, pp. 1154-1160, Nov. 2011. Article (CrossRef Link)

[10] P. Besl and N. McKay, "A Method for Registration of 3-D Shapes," *IEEE Trans. Patten Analysis and Machine Intelligence*, vol. 14, pp. 239-256, 1992. Article (CrossRef Link)

[11] R. Toldo, A. Beinat, and F. Crosilla, "Global registration of multiple point clouds embedding the generalized procrustes analysis into an ICP framework," in *3DPVT 2010 Conf., Paris*, May 17-20, 2010. Article (CrossRef Link)

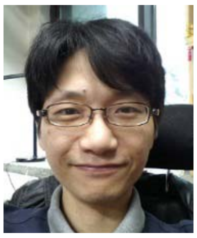
[12] S. Bouaziz, A. Tagliasacchi, and M. Pauly, "Sparse Iterative Closest Point," *Computer Graphics Forum*, vol. 32, no. 5, pp. 1–11, 2013. Article (CrossRef Link)

[13] K. Shoemake, "Animating rotation with quaternion curves," in *Proc. of the SIGGRAPH* '85, 1985, pp. 245-254. Article (CrossRef Link)

[14] NVidia CUDA Article (CrossRef Link)

[15] CGAL Article (CrossRef Link)

[16] Sparse ICP Article (CrossRef Link)



Bumsik Yoon received his BSc and MSc in Electrical Engineering from Yonsei University, Korea, in 1997 and 2000, respectively. He is a senior researcher at Samsung Electronics. Currently, he is pursuing his PhD at Hanyang University, Korea. His research interests include 3D reconstruction, pedestrian detection, time-of-flight and computer graphics.

Kunwoo Choi received his BSc in Electronics Engineering at Konkuk University, Korea, in 2014. He is currently pursuing an MSc in Electronics and Computer Engineering at Hanyang University. His research interests include depth acquisition and vehicle vision systems.

Moonsu Ra received his BSc and MSc at Hanyang University, Korea, in 2011 and 2013, respectively. He is currently pursuing his PhD at the same university. His research interests include visual surveillance, face tracking and identification, and video synopsis.

Whoi-Yul Kim received his PhD in Electronics Engineering from Purdue University, West Lafayette, Indiana, in 1989. From 1989 to 1994, he was with the Erick Johansson School of Engineering and Computer Science at the University of Texas at Dallas. He joined Hanyang University in 1994, where he is now a professor in the Department of Electronics and Computer Engineering. His research interests include visual surveillance, face tracking and identification, and video synopsis.