Vis Comput (2016) 32:955–965
DOI 10.1007/s00371-016-1249-5

ORIGINAL ARTICLE

Texture map generation for 3D reconstructed scenes

Junho Jeon[1] · Yeongyu Jung[1] · Haejoon Kim[1] · Seungyong Lee[1]

Published online: 6 May 2016

Abstract We present a novel method for generating texture maps for 3D geometric models reconstructed using consumer RGB-D sensors. Our method generates a texture map for a simplified 3D mesh of the reconstructed scene using spatially and temporally sub-sampled key frames of the input RGB stream. We acquire an accurate texture map by optimizing the texture coordinates of the 3D model to maximize the photometric consistency among multiple key frames. We show that the optimization can be performed efficiently using GPU by exploiting the locality of texture coordinate manipulation. Experimental results demonstrate that our method can generate a texture map in a few tens of seconds for a large 3D model, such as a whole room.

Keywords 3D reconstruction · Texture mapping · RGB-D images · Photometric consistency optimization

1 Introduction

Nowadays RGB-D cameras, such as Microsoft Kinect, have become widely available. Various researches on 3D reconstruction based on RGB-D images, e.g., KinectFusion and its variations [10–12,15], enabled 3D navigation of a scene by rendering the reconstructed 3D model from desirable viewpoints. However, these reconstructed 3D models are not yet popularly used in applications, such as virtual reality and augmented reality, due to the lack of accurate color information. Most reconstruction methods so far recover the color information by blending the corresponding color values in different input images. Imprecisely estimated camera poses and lens distortions can cause misalignments between images, and consequently this simple blending may induce blurring and ghosting artifacts in the blended surface colors and diminish the quality of the rendering results.

Recently, Zhou and Koltun [17] proposed an optimization-based approach for mapping color images onto a 3D geometric model reconstructed using a consumer RGB-D sensor. They optimized the camera poses for all color images and applied non-rigid transforms to color images to enhance the alignments with the 3D model. In their approach, the 3D reconstructed model is rendered using the *vertex colors* computed by weighted averages of aligned color values. To represent the detailed color information of a large reconstructed scene, it would require a large number of vertices that introduce storage and rendering overheads. In addition, their alternating optimization takes several minutes for a small object, and the approach is not much scalable to handle large indoor scenes.

In this paper, we propose a novel approach based on *texture mapping* for mapping color images onto a 3D reconstructed scene. Differently from previous methods [11,17] that use voxel or vertex colors to render 3D reconstructed models, our approach uses an accurately recovered texture map for rendering. The usage of texture mapping enables a simpler mesh to be used for reconstructing detailed color information of a large scene, such as a whole room.

However, it is not straightforward to generate a texture map without blurring and ghosting artifacts from input color images, as the color images and the reconstructed 3D model should be precisely aligned to produce such a texture map.

Electronic supplementary material The online version of this article (doi:10.1007/s00371-016-1249-5) contains supplementary material, which is available to authorized users.

✉ Seungyong Lee
leesy@postech.ac.kr

[1] Department of Computer Science and Engineering, POSTECH, Pohang, Korea

Although it could be possible to generate a texture map from vertex colors, such an approach would need optimized colors for a huge number of vertices to handle a large scene. To efficiently generate an accurate texture map usable for rendering a large 3D reconstructed model, we present a novel framework for mapping color images onto a 3D model with the following technical contributions:

– Texture map generation method that maximizes the photometric consistency of input images in the global texture map.
– Efficient optimization of the texture map generation method based on a parallel Gauss–Newton solver running on GPU.
– Spatiotemporal adaptive sampling of input images that reduces the redundant information and motion blurs for faster processing and sharper texture maps.

Experimental results on various RGB-D image data demonstrate that the proposed method reconstructs high-quality texture maps within a few seconds for small objects and within practical running times for large indoor scenes.

2 Related work

To generate 3D models for real objects or scenes from given images, 3D reconstruction has been widely researched in computer graphics and vision [1,4,14]. As depth cameras have become popular, recent developments focus on reconstruction using RGB-D images. *KinectFusion* proposed by Newcombe et al. [10] is one of the pioneering works that reconstructs the 3D geometry using a volumetric representation. Based on *KinectFusion*, several extensions [10,11,15, 16] have been proposed. Among them, Niener et al. [11] proposed a large-scale indoor scene reconstruction method using a hash-based geometric representation. In our approach, we use their method in the model reconstruction step to generate a 3D mesh with estimation of the camera poses of input color images. We also use the 3D models and corresponding camera poses reconstructed by [16] for the experiments.

Beyond the 3D geometry reconstruction, mapping the color information from given multiple images onto the reconstructed model has not been heavily investigated yet. In most 3D reconstruction methods [10,11], estimated relative camera poses between input images are used to average the pixel colors for object points. Hence, imprecise alignments between color images cause blurring and ghosting artifacts, which lower the rendering quality. Our texture coordinate optimization resolves the misalignments by imposing photometric consistency among color images projected on the reconstructed model.

The color map optimization method proposed by Zhou and Koltun [17] is one of the state-of-the-art color reconstruction techniques. After reconstructing a 3D model from a depth image stream, they optimize the alignments among color images with respect to every vertex in the model. They also optimize a non-rigid deformation function for each image to correct distortions due to imprecise 3D reconstruction. Although their method can precisely refine the color mapping for a reconstructed 3D object, it takes several hundreds of seconds for a small object due to its time-consuming nonlinear optimization. By applying texture mapping to a 3D model and exploiting the locality of texture coordinate optimization, our method can generate a high-quality texture map within a few seconds for a small object.

3 Overview

3.1 Overall process

Figure 1 shows the overall process of our approach to generate a texture map for a reconstructed 3D model. We use a depth and color image stream as the input. For a given input stream, we reconstruct a geometric model using a voxel-based 3D reconstruction method, and the reconstructed model is simplified using a mesh simplification method. We then select key frames from the color image stream using spatiotemporal adaptive sampling to reduce the processing time and to increase the quality of the generated texture map.

To generate a global texture map, we first estimate multiple sub-textures for each face by projecting its vertices onto the selected key frames in which the face is visible. The global texture map for a face is determined by weighted blending of the multiple sub-textures. The initial sub-textures can be photometrically inconsistent to each other because of imprecisely estimated camera poses and possible distortions in the key frames. We refine the sub-textures of faces by optimizing an objective function of sub-texture coordinates to maximize the photometric consistency. This optimization enables us to obtain a precise global texture map that can be rendered efficiently with the simplified mesh.

3.2 Preprocessing of input data

Geometric model reconstruction For a given depth and color image stream, we first reconstruct a 3D geometric model with a triangular mesh representation. Any 3D reconstruction method can be used for this step. In this paper, we use [11,16] to obtain a 3D model from a given RGB-D image stream. For [16], we used the reconstructed 3D models provided by the authors for experiments.

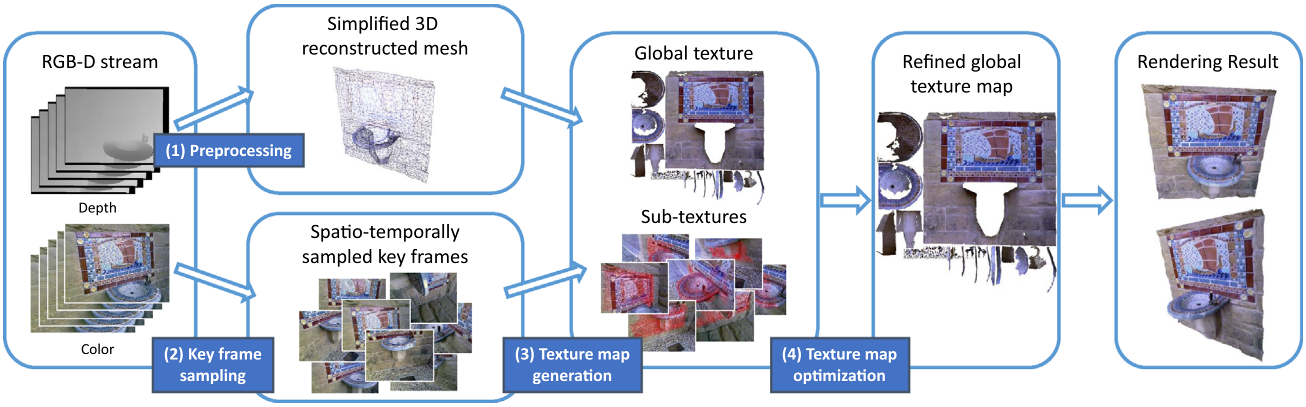

RGB-D stream
Depth
Color
(1) Preprocessing
Simplified 3D reconstructed mesh
(2) Key frame sampling
Spatio-temporally sampled key frames
(3) Texture map generation
Global texture
Sub-textures
(4) Texture map optimization
Refined global texture map
Rendering Result

Fig. 1 The overall process of our texture map generation method for a 3D reconstructed scene. Input *color images* are mapped and blended on a simplified version of the 3D reconstructed mesh to produce a global texture map. Our method refines the global texture map by optimizing the texture coordinates of multiple sub-textures mapped onto each face of the simplified 3D mesh

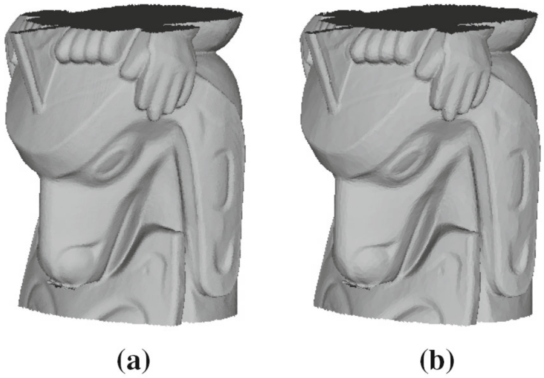


Fig. 2 Mesh simplification example. The number of faces is reduced from **a** 460 K to **b** 23 K

Model simplification Even for a small scene, the initial mesh $\mathbf{M}^0$ usually consists of millions of vertices and faces, which are too heavy for the remaining process. In addition, rendering with texture mapping becomes inefficient when the model contains too large number of faces. To address these issues, we apply a mesh simplification method with quadric error metric [6] to $\mathbf{M}^0$ for reducing the number of faces while preserving the shape. In Fig. 2, we reduced the number of faces significantly, up to 5 % of the original, but geometric details are still almost preserved. The simplified mesh is denoted by $\mathbf{M}$, which is the input of the remaining process.

4 Spatiotemporal key frame sampling

The length of an input RGB-D stream can vary from several hundreds to thousands of frames, according to the scale of the target scene to be reconstructed. Such a long stream contains a lot of redundant data, most of which are less useful for texture map generation. Also, the color image stream captured by a handheld camera suffers from motion blurs that would lower the quality of the generated texture map. By sampling the input images according to the uniqueness and quality, we can accelerate the texture generation process and obtain a better quality texture map. We call this key frame selection process *spatio-temporal sampling*.

4.1 Temporal sampling using blurriness

We first sample the images in the temporal domain by selecting relatively less blurred frames. We use the method of Crete et al. [5] to measure the blurriness of input frames. Similarly to Zhou and Koltun [17], we select key frames one by one with the smallest blurriness values. Specifically, we first check the blurriness of $\sigma_{\max}$ frames from the beginning of the stream and choose the key frame with the smallest blurriness. We then skip $\sigma_{\min}$ frames from the last selected key frame to avoid redundant frames. Then again, we check $\sigma_{\max}$ frames from the last skipped frame to select the next key frame and repeat the process. We used $\sigma_{\min} = 5$ and $\sigma_{\max} = 30$ for all experiments in this paper.

Note that we use a shorter time interval (several frames) for key frame selection than [17], which uses 1–5 s for the interval. Our method aims to handle reconstruction of indoor scenes rather than small objects. In a stream capturing an indoor scene with clutters, small parts of the scene could be captured only in small portions of the stream. If a long time interval is used for key frame selection, we can easily miss the details of the scene. Therefore, we use a shorter time interval to select as many high-quality key frames as possible, and filter out the redundant frames using spatial sampling.

4.2 Spatial sampling using uniqueness

After sampling key frames in the temporal domain with blurriness, we perform sampling in the spatial domain. To reduce

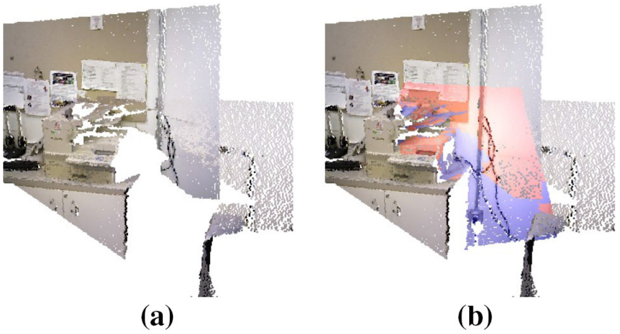


Fig. 3 Frame uniqueness measurement. **a** Temporally sampled key frames $\mathbf{I}_t$, **b** unique regions (*blue points*) and overlapping regions (*red points*) of a frame I against the frame set $\mathbf{I}_t$

redundant information, we sample key frames that contain unique information which is distinguishable from others. To measure the uniqueness of a frame I, we first define the uniqueness of a pixel p in I against a set of temporally sampled key frames $\mathbf{I}_t$ as:

$$q_I(p) = \begin{cases} 1, & \text{if } |z_I(p) - z_{I'}(p')| > \sigma_s \quad \forall I' \in \mathbf{I}_t, \\ 0, & \text{otherwise}, \end{cases} \tag{1}$$

where p' is the 2D position when p in I has been projected onto I', $z_I(p)$ is the depth value of pixel p in I, and σ_s is a user-specified overlapping threshold. That is, a depth pixel is unique when it does not overlap with any other depth pixels in $\mathbf{I}_t$. Figure 3 shows examples of unique and overlapping depth pixels. Based on the pixel uniqueness, the uniqueness of frame I is evaluated as:

$$Q(I) = \frac{1}{|I|} \sum_{p \in I} q_I(p), \tag{2}$$

where $|I|$ is the number of pixels in I.

Evaluating the uniqueness for all temporally sampled key frames would be time consuming, as we should project every pixel in each key frame onto all other key frames. As the number of key frames increases, computation time for evaluating the uniqueness increases accordingly. For a large-scale scene that consists of thousands of images, it may take few hours. To reduce the computation time, we can use two types of acceleration: downsampling and parallelization.

We can approximate the frame uniqueness by downsampling an image into uniform grids and only using the uniqueness of representative points of the grids. We use the average of all depth pixels in each grid as the representative point. In our implementation, we use 10 × 10 uniform grids. Since each grid can cover a large region of the scene, we use the overlapping threshold $\sigma_s = 20\,\text{cm}$ for all results in this paper. While preserving the quality, this approximation gives huge acceleration compared to full image processing.

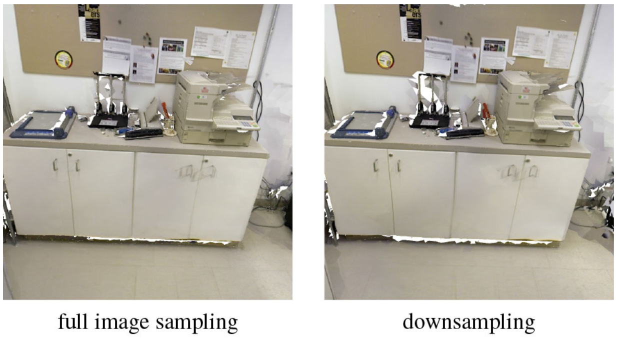


Fig. 4 Results with/without frame uniqueness approximation. The quality of the final rendering result is not much affected

Figure 4 shows that the quality of the final rendering result is not much affected by this approximation. Another option for acceleration is to use GPU for parallel implementation of the full image sampling. Our CUDA implementation of the frame uniqueness computation based on full images runs twice as fast as the CPU-based downsampling version.

After calculating the uniqueness of all key frames, we remove the key frames with the smallest uniqueness values one by one while updating the uniqueness of neighboring key frames after each removal. Two images are neighbors when they have overlapped representative points. The removal process continues until the smallest uniqueness becomes large enough, resulting in key frames that have enough unique information among each other. This spatial sampling of key frames can be efficiently performed by maintaining a priority queue for the uniqueness values.

As a result of temporal and spatial sampling, the size of the input stream is significantly reduced from thousands to tens or hundreds of frames. Despite this aggressive sampling, the quality of final rendering results are not much degraded, as demonstrated in Fig. 10.

5 Texture map generation

After we have chosen a set of key frames using spatiotemporal sampling, we use them to generate a global texture map for the simplified 3D mesh. This process consists of two steps, global texture map generation and texture coordinate optimization. The first step generates a global texture map by UV parameterization of the 3D mesh and blending of key frame images. The second step refines the generated global texture map by optimizing an objective function that imposes photometric consistency on blended images.

5.1 Global texture map generation

We generate a single global texture map for the simplified 3D mesh $\mathbf{M}$ by blending the key frames $\{I_i\}$. Let $\{v_i\}$ and

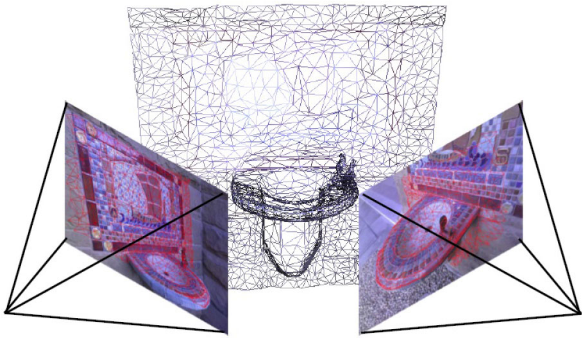


Fig. 5 Sub-texture coordinate estimation. Each vertex is projected onto the sub-textures (key frames) to estimate the corresponding 2D sub-texture coordinates

$\{f_i\}$ denote the vertices and faces of **M**, respectively. First, for each face f, we find a set of key frames in which f is visible. We call this set *sub-textures* of f. At the same time, as Fig. 5 shows, we estimate the 2D sub-texture coordinates of vertices of each face f by back-projecting the vertices onto each sub-texture of f. As each face has multiple sub-textures, a vertex v_i of face f has multiple sub-texture coordinates u_{ij} that are the projected coordinates of vertex v_i onto sub-textures I_j. Then, each sub-texture can be mapped onto the mesh **M** using the sub-texture coordinates.

After we have estimated the sub-texture coordinates of all vertices, we generate a global texture map by blending the sub-textures of each face. Blending weight of a sub-texture I_j for vertex v_i is defined as $w_{ij}^b = \mu \cos(\theta)$, where θ is the angle between the surface normal of v_i and the view direction of the camera for I_j, and μ is a smooth weighting function that gives a higher weight when the sub-texture coordinates u_{ij} is close to the image center of I_j.

After the blending, the blended sub-texture of each face is copied onto the global texture map, which is a huge image that contains all blended sub-textures. As the reconstructed 3D mesh does not have its own texture mapping information, we should generate texture coordinates of all vertices using UV parameterization. In our method, texture coordinates are determined using the least squares conformal map generation method [8] that minimizes angle deformations of surface triangles during the parameterization. Figure 6 shows the generated global texture map and sub-textures that have been blended for the *fountain* dataset.

Although we sampled the key frames considering spatial redundancy, there still can be too many key frames from which a face is visible. For computational efficiency, we limit the number of sub-textures for each face. We choose the top k key frames that have the highest blending weights w_{ij}^b for the sub-textures. We used $k = 10$.

The initially generated global texture map may suffer from blur and ghosting artifacts, which are caused by an imprecisely reconstructed 3D model and optical distortions in color images. To eliminate these artifacts and enhance the rendering quality, we optimize the global texture map by updating the sub-texture coordinates of vertices to maximize the photometric consistency among the overlapping sub-textures.

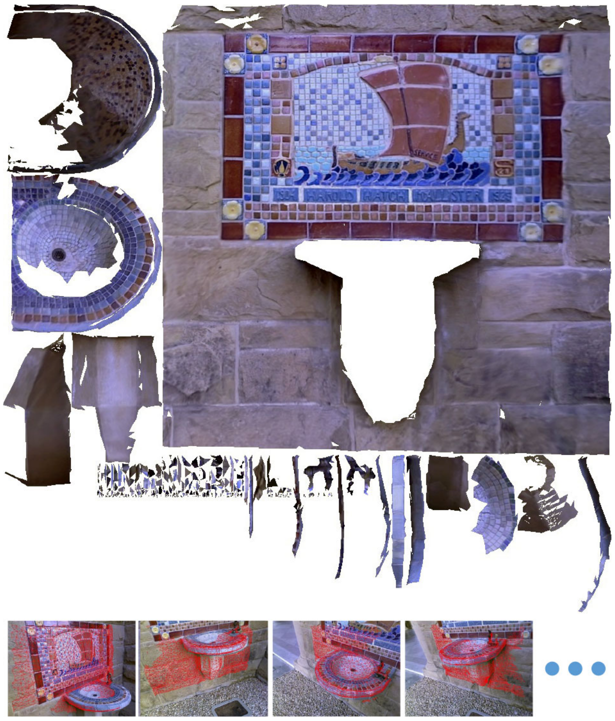


Fig. 6 Generated global texture of the *fountain* dataset and some of sub-textures blended for the global texture

5.2 Global texture map optimization

Differently from Zhou and Koltun's method [17] that optimizes the non-rigid transforms of key frames, we directly optimize the sub-texture coordinates of each vertex of the simplified mesh **M**. Since **M** contains relatively larger faces than the initial dense mesh $\mathbf{M^0}$, measuring the photometric consistency at only vertices would not suffice to refine the global texture map. To consider the consistency over the entire mesh, we choose a few sample points on each face. Let $S_i(f, w)$ denote the sub-texture coordinates on I_i of a sample point from face f, which are determined by the linear combination of the sub-texture coordinates of the three vertices of f with a combination weight w.

Our objective is to minimize the inconsistency among the sub-textures by modifying the sub-texture coordinates of vertices. Let **u** denote the concatenation of all sub-texture coordinates that we want to optimize. Also, let $\mathbf{F} = \{f_i\}$ and $\mathbf{V} = \{v_i\}$. We can quantify the sum of photometric inconsistency of all sample points on **F** as:

$$E(\mathbf{u}) = \sum_{f \in \mathbf{F}} C(f), \qquad (3)$$

where $C(f)$ is inconsistency among sample points on the sub-textures $\mathbf{I}_f$ belonging to a face f. We compute $C(f)$ as the inconsistency sum of image pairs in $\mathbf{I}_f$:

$$C(f) = \sum_{I_i, I_j \in \mathbf{I}_f} M(f, I_i, I_j), \tag{4}$$

where $M(f, I_i, I_j)$ is a function that measures the inconsistency of sample points on f between sub-textures I_i and I_j. It is defined as:

$$M(f, I_i, I_j) = \sum_{w \in \mathbf{w}} \left(\Gamma_i(S_i(f, w)) - \Gamma_j(S_j(f, w))\right)^2, \tag{5}$$

where $\Gamma_i(u)$ is the color value at the 2D texture coordinates u on I_i, and $\mathbf{w}$ is the set of combination weights for the sampled points on f. As $S(f, w)$ is a function of sub-texture coordinates of vertices, we can obtain optimal sub-texture coordinates $\mathbf{u}$ by minimizing Eq. (3).

To efficiently minimize the objective function in Eq. (3), we introduce two modifications. First, we re-arrange the objective with respect to the vertices rather than faces. Let $\mathbf{F}_v$ be the 1-ring neighbor faces of a vertex v, which is the set of faces that contain v as one of their three vertices. Then the modified objective can be written as:

$$E(\mathbf{u}) = \frac{1}{3} \sum_{v \in \mathbf{V}} \sum_{f \in \mathbf{F}_v} C(f). \tag{6}$$

Each face consists of three vertices, so Eqs. (3) and (6) are equivalent.

Second, we introduce an auxiliary variable $P(f, w)$, which is the proxy color of a sample point on f with a combination weight w. Then we re-formulate Eqs. (4) and (5) to:

$$C(f) = \sum_{I_i \in \mathbf{I}_f} M(f, I_i), \tag{7}$$

$$M(f, I_i) = \sum_{w \in \mathbf{w}} \left(\Gamma_i(S_i(f, w)) - P(f, w)\right)^2. \tag{8}$$

As a result, our objective can be written as $E(\mathbf{u}, \mathbf{P})$, where $\mathbf{P}$ is the concatenated vector of all auxiliary variables $P(f, w)$. For simplicity, in the optimization process, we use grayscale values of the key frame images, so $\Gamma_i(S_i(f, w))$ and $P(f, w)$ are scalar values.

5.3 GPU-based alternating solver

Our objective $E(\mathbf{u}, \mathbf{P})$ is a non-linear least squares function of sub-texture coordinates $\mathbf{u}$ and auxiliary variables $\mathbf{P}$, which can be minimized using the Gauss–Newton method. Similar to Zhou and Koltun [17], we can also use alternating optimizations that optimize $\mathbf{u}$ and $\mathbf{P}$ separately while fixing the other. However, for a large reconstructed model such as a whole room, we need more drastic acceleration because the number of parameters of the objective, even for the simplified mesh, can be huge, up to $k|\mathbf{V}|$. To optimize the objective with such a large number of parameters in practical computation time, we optimize the objective function in parallel by exploiting the locality of the problem.

Our alternating optimization consists of two stages. At the first stage, we optimize $\mathbf{P}$ while $\mathbf{u}$ is fixed. Then it can be easily shown that the objective is minimized when

$$P(f, w) = \frac{1}{|\mathbf{I}_f|} \sum_{I_i \in \mathbf{I}_f} \Gamma_i(S_i(f, w)), \tag{9}$$

for all $f \in \mathbf{F}$ and $w \in \mathbf{w}$. The values of $P(f, w)$ are independent of each other, so they can be computed quickly in parallel.

At the second stage, we optimize $\mathbf{u}$, while $\mathbf{P}$ is fixed. This reduces to a non-linear least square problem as follows:

$$E(\mathbf{u}) = \frac{1}{3} \sum_{v \in \mathbf{V}} \sum_{f \in \mathbf{F}_v} \sum_{I_i \in \mathbf{I}_f} \sum_{w \in \mathbf{w}} r(f, I_i, w)^2, \tag{10}$$

where $r(f, I_i, w)$ is the residual considered in Eq. (8):

$$r(f, I_i, w) = \Gamma_i(S_i(f, w)) - P(f, w). \tag{11}$$

Then single Gauss–Newton update is defined as:

$$\mathbf{u}^{a+1} = \mathbf{u}^a + \Delta\mathbf{u}, \tag{12}$$

where $\Delta\mathbf{u}$ is computed as the solution of the equation:

$$J(\mathbf{u}^a)^T J(\mathbf{u}^a) \Delta\mathbf{u} = -J(\mathbf{u}^a)^T R(\mathbf{u}^a). \tag{13}$$

$R$is the residual vector that is the concatenation of $r(f, I_i, w)$ for all f, I_i, and $w.J$ is the Jacobian of R.

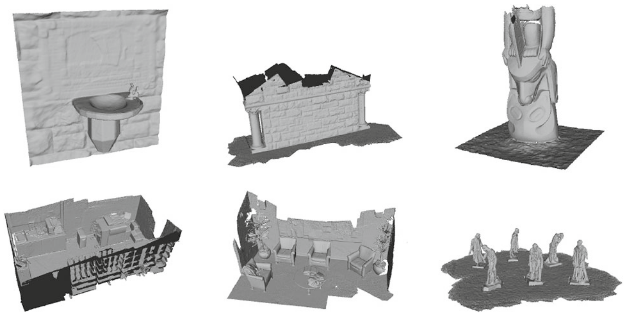
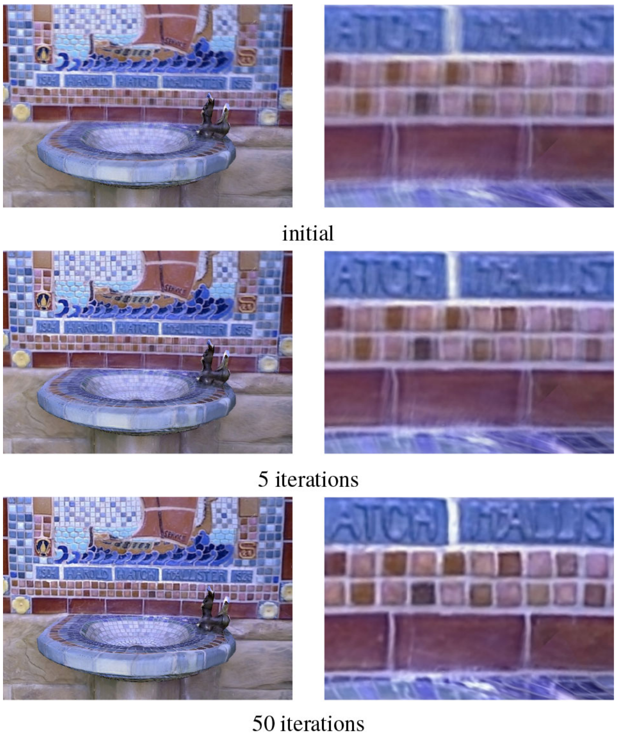


Fig. 7 3D reconstructed models used in our experiments: from *top left, fountain, stonewall, totempole, copyroom, lounge, burghers*. The models were reconstructed using the methods of Zhou and Koltun [16, 17]

Although the Jacobian matrix J is quite sparse, its size is vast for a large 3D model. For example, the reconstructed 3D model of the *lounge* dataset in Sect. 6 consists of more than 2M (millions) vertices originally. Even if we apply mesh simplification to the model, the number of vertices should be more than 0.1 M for preserving the details. Then the number of parameters in the optimization goes up to 1 M and the dimension of the Jacobian matrix becomes 1 M × 1 M. Solving such a large linear system on GPU is non-trivial, and the optimization is not readily parallelizable.

To parallelize the optimization, we exploit the locality of the problem. As in Sect. 5.2, let $\mathbf{F}_v$ be the one-ring neighborhood of a vertex v. A change of the sub-texture coordinates of v only affects the consistencies, i.e., residuals, of sample points on the faces in $\mathbf{F}_v$. In other words, the consistency among sub-textures around v is only determined by $\mathbf{F}_v$. Thus, we can subdivide the entire Gauss–Newton update problem into smaller independent sub-problems that only update the sub-texture coordinates of single vertices. These small sub-problems can be solved by performing a parallel variant of *Schwarz alternating method* [7] that updates the inner variables (texture coordinates of v) using a Gauss–Newton update while keeping the boundary variables (texture coordinates of one-ring neighbor vertices of v).

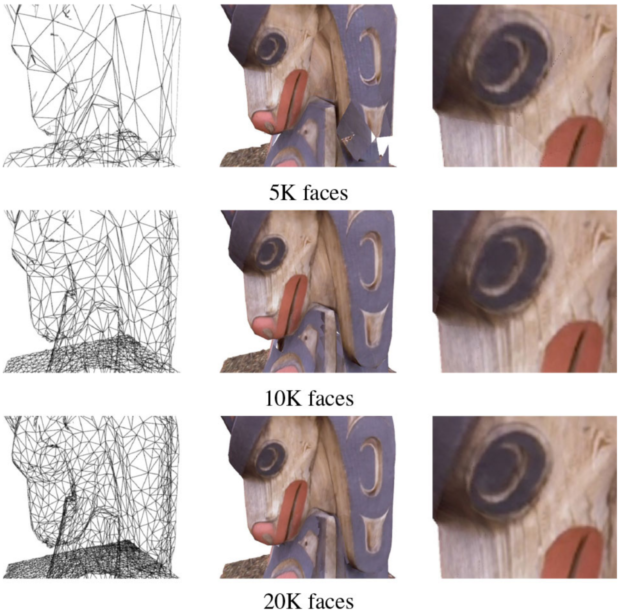
Specifically, the parallel Gauss–Newton update works as follows. As described above, each Gauss–Newton update needs to solve the linear system which is a function of the residuals. If we ignore all the residuals that are independent


initial

5 iterations

50 iterations

Fig. 8 Progress of the iterative texture optimization on a cropped region of the *fountain* model. As the objective function is minimized, the photometric consistency of the generated texture map is improved drastically. It is best viewed on a *color* monitor


5K faces

10K faces

20K faces

Fig. 9 Rendering results of the *totempole* model simplified to different numbers of faces. Even with only 10 K faces, the rendering quality is not much degraded, while the optimization takes less than 3 s. Note that the original mesh consists of more than 1 M faces

Table 1 Test dataset statistics and the computation time of our method

Model	# of images	# of key frames	# of original faces	# of simplified faces	Time for optimization (s)
Fountain	1086	27	532,806	10,000	2.6
Totempole	2700	103	1,285,993	10,000	2.5
Stonewall	2700	113	4,345,379	65,000	9.5
Lounge	3000	106	3,150,436	130,000	16
Copyroom	5490	155	5,062,748	130,000	18
Burghers	11,230	319	6,858,620	195,000	31

For a large-scale model, the proposed GPU-based alternating optimization takes only few tens of seconds to obtain a photometrically enhanced global texture map of a vertex v and assume that the sub-texture coordinates of vertices other than v in $\mathbf{F}_v$ are fixed, then the linear system in Eq. (13) becomes very small, and evaluating the residual and numerical calculation of the inverse of a 2×2 matrix is only required. With this simplification, we can use single kernel threads on GPU to perform parallel Gauss–Newton updates for the sub-texture coordinates of vertices. To propagate the updated sub-texture coordinates to the one-ring neighborhood aggressively, we update $\mathbf{u}$ twice for each $\mathbf{P}$ update. In Sect. 6, the experimental results show the effectiveness of our GPU-based alternating solver, which can handle optimization for a large 3D model within tens of seconds.

6 Experimental results

We performed various experiments to evaluate the proposed method. We tested our method on several 3D reconstructed models with RGB-D streams (Fig. 7), provided by Zhou and Koltun [16,17], which were taken using an Asus Xtion Pro Live RGB-D camera. Except the *fountain* data, we used 3D meshes and camera trajectories that were estimated by [16]. For the *fountain* data, we used voxel hash-based reconstruction [11] to reconstruct the 3D model and estimated the camera trajectories from the given RGB-D images. All experiments were performed on a PC with an Intel i7 4.0 GHz CPU, 16 GB RAM and Nvidia GeForce GTX TITAN X GPU. The implementations of algorithms [6,8] in MeshLab [3] and Blender [2] were used for mesh simplification and UV parameterization of a simplified mesh, respectively.

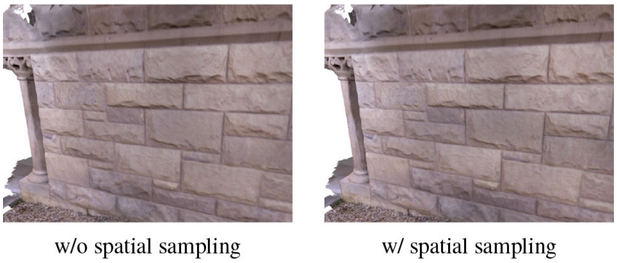

w/o spatial sampling w/ spatial sampling

Fig. 10 Rendering results with/without spatial sampling. Spatial sampling removes a half of key frames, which are redundant, preserving the rendering quality

Timing data Our method refines the global texture map using iterative optimization. Figure 8 shows that the rendering results are progressively refined through the optimization of the sub-texture coordinates. The blurry texture map becomes sharp in only few tens of iterations. We used 100 iterations for all experiments in this paper. Table 1 shows computation time and other statistics of our method. Our spatiotemporal key frame sampling and the initial global texture generation takes a few seconds, depending on the size of the image stream. Texture map optimization takes several to tens of seconds, which is much faster than that of Zhou and Koltun [17]. For example, for the *fountain* model, Zhou and Koltun [17] took more than 200 s according to their paper, but our optimization only takes 2.6 s, taking 26 ms per iteration.

Scalability With the GPU-based alternating solver, our approach has much higher scalability than that of [17]. Zhou and Koltun [17] formulated the optimization problem as n independent linear systems with 720 variables, where n is the number of key frames. In that case, different linear systems cannot be solved in parallel on a GPU, because solving each system would need multiple kernel threads. In contrast, exploiting the locality, our GPU-based alternating solver sep-

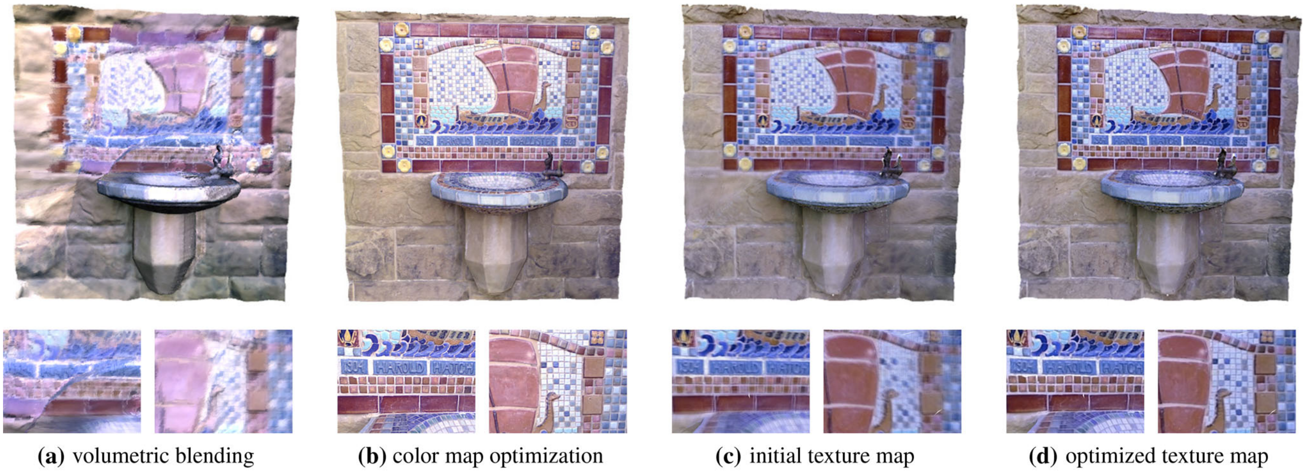

(a) volumetric blending **(b)** color map optimization **(c)** initial texture map **(d)** optimized texture map

Fig. 11 Rendering result of the *fountain* model with our generated texture map. We compare the result with the volumetric blending approach [11] and the *color map* optimization [17]. Note that the overall difference of color balances between [17] and ours has come from the difference in the input image streams

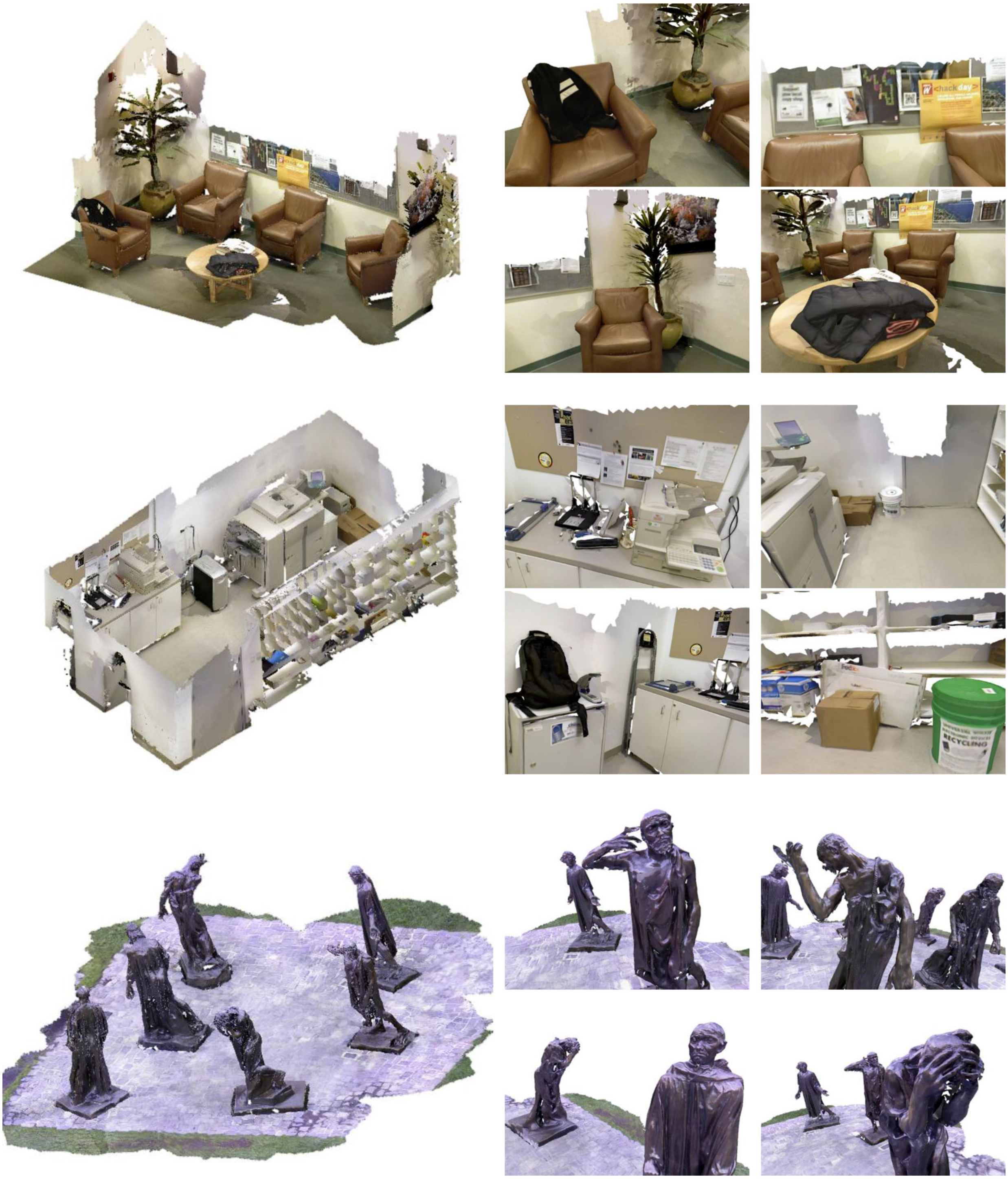


Fig. 12 Texture map generation and optimization results for large-scale 3D reconstructed models (*lounge, copyroom, burghers*). White holes in the rendering results have come from the missing geometry of the reconstructed models

arates the problem into m independent linear systems with only two variables, where m is the number of vertices of the simplified mesh. Many of these small linear systems can be solved in parallel on a GPU, as each system can be handled by a single kernel thread with few arithmetic operations.

Mesh simplification We first evaluate the robustness of our method against mesh simplification in terms of the rendering quality. As described in Sect. 5.3, the processing time of texture coordinate optimization depends on the number of vertices in the simplified mesh, so we can achieve more acceleration with aggressive mesh simplification. Figure 9 demonstrates that the rendering quality is still satisfactory even when the number of faces is extremely reduced to 1 % of the original.

Adaptive sampling We also evaluate the effectiveness of our spatiotemporal adaptive sampling. Temporal sampling based on the blurriness reduces the number of key frames depending on the length of the input stream. Then the spatial sampling based on uniqueness reduces the key frame set based on the scale of the scene. It shortens the processing time of texture map optimization by eliminating redundant images which are not necessarily useful for reconstructing the 3D model. Figure 10 shows the rendering results when only temporal sampling is applied and when both temporal and spatial samplings are applied. By performing spatial sampling, we reduced the number of sampled key frames from 214 to 113 while preserving the rendering quality. Note that the images participating in the sub-texture blending could have been changed due to the spatial sampling, causing some color differences between the rendering results.

Visual comparison For the *fountain* model, we compared our result with Zhou and Koltun [17] using the reconstructed mesh of [17] provided by the authors. Figure 11 shows a visual comparison. The result in [17] was generated from a high-resolution (1920 ×1080) color image stream, which was not available to us. Instead, we used a low-resolution (640 × 480) stream provided by the authors to generate our global texture map. Nevertheless, the rendering results are quite comparable, while our processing time is about 100× faster. Note that the color balance of the rendering result in [17] differs from ours, because the input color streams are not identical. We applied auto white balancing to our input color stream to compensate for the difference.

Large-scale model Due to the adaptive key frame sampling and GPU-based alternating optimization, our approach is scalable to handle large-scale 3D reconstruction. We demonstrate the scalability of our method by generating texture maps for reconstructed 3D indoor scenes. We tested three models, *copyroom, lounge, burghers*, which consist of more than 5 M faces on average. Each model is simplified to 130 or 195 K faces based on the geometric complexity. Figure 12 shows several rendering results from different viewpoints. These examples demonstrate that our method can be used to generate precise texture maps needed for visualizing large-scale 3D reconstructed indoor scenes.

7 Conclusions

Differently from previous works based on voxel or vertex colors, this paper proposed a texture mapping-based approach for representing and rendering color information of a 3D reconstructed model. Reconstructed geometric models, especially indoor scenes, equipped with precisely generated texture maps can be used as 3D model contents for various applications, such as virtual reality and 3D fabrication.

Limitation and future work Our algorithm generates a single global texture map for a whole 3D reconstructed scene. When the algorithm is applied to a large-scale scene, the required size of the global texture map could become too large. Subdividing a large model into small ones and generating separate texture maps can resolve this issue. Our texture map optimization method only considers local information (one-ring neighborhood) of each vertex to enhance the photometric consistency. Exploiting this locality enables our method to work very fast with parallel implementation, but prohibits the method from working with a dense mesh where local information is limited. As shown in Fig. 6, our global texture map currently contains lots of holes on which no vertices are assigned. Advanced mesh parameterization methods [9,13] would be useful to resolve this problem.

Acknowledgments This work was supported by the National Research Foundation of Korea (NRF) Grant (NRF-2014R1A2A1A11052779) and Institute for Information and Communications Technology Promotion (IITP) Grant (R0126-16-1078), both funded by the Korea government (MSIP).

References

1. Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S.M., Szeliski, R.: Building rome in a day. Commun. ACM **54**(10), 105–112 (2011)
2. Blender foundation: blender. https://www.blender.org. Accessed Jan 2016
3. Cignoni, P., Corsini, M., Ranzuglia, G.: MeshLab: an open-source 3D mesh processing system. Ercim News **73**(45–46), 6 (2008)
4. Crandall, D., Owens, A., Snavely, N., Huttenlocher, D.: Discrete-continuous optimization for large-scale structure from motion. In: 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3001–3008. IEEE (2011)
5. Crété-Roffet, F., Dolmiere, T., Ladret, P., Nicolas, M.: The blur effect: perception and estimation with a new no-reference perceptual blur metric. In: SPIE Electronic Imaging Symposium Conf Human Vision and Electronic Imaging, vol. 12, pp. EI–6492 (2007)
6. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, pp. 209–216. ACM Press/Addison-Wesley Publishing Co., New York (1997)
7. Hazewinkel, M. (ed.): Encyclopaedia of Mathematics (Set). Springer, The Netherlands (1994)
8. Lévy, B., Petitjean, S., Ray, N., Maillot, J.: Least squares conformal maps for automatic texture atlas generation. ACM Trans. Graph. (TOG) **21**(3), 362–371 (2002)
9. Liu, L., Zhang, L., Xu, Y., Gotsman, C., Gortler, S.J.: A local/global approach to mesh parameterization. In: Computer Graphics Forum, vol. 27, pp. 1495–1504. Wiley Online Library, New York (2008)
10. Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohi, P., Shotton, J., Hodges, S., Fitzgibbon,

A.: Kinectfusion: real-time dense surface mapping and tracking. In: 2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), pp. 127–136. IEEE (2011)
11. Niener, M., Zollhfer, M., Izadi, S., Stamminger, M.: Real-time 3D reconstruction at scale using voxel hashing. ACM Trans. Graph. (TOG) **32**(6), 169 (2013)
12. Roth, H., Vona, M.: Moving volume kinectfusion. In: BMVC, pp. 1–11 (2012)
13. Smith, J., Schaefer, S.: Bijective parameterization with free boundaries. ACM Trans. Graph. (TOG) **34**(4), 70 (2015)
14. Tomasi, C., Kanade, T.: Shape and motion from image streams under orthography: a factorization method. Int. J. Comput. Vis. **9**(2), 137–154 (1992)
15. Whelan, T., Johannsson, H., Kaess, M., Leonard, J.J., McDonald, J.: Robust tracking for real-time dense rgb-d mapping with kintinuous. Technical Report MIT-CSAIL-TR-2012-031, CSAIL, MIT (2012)
16. Zhou, Q.Y., Koltun, V.: Dense scene reconstruction with points of interest. ACM Trans. Graph. (TOG) **32**(4), 112 (2013)
17. Zhou, Q.Y., Koltun, V.: Color map optimization for 3D reconstruction with consumer depth cameras. ACM Trans. Graph. (TOG) **33**(4), 155 (2014)

Junho Jeon received his B.S. degree from the Pohang University of Science and Technology (POSTECH), Pohang, South Korea, in 2012. He is currently a Ph.D. student in Computer science and engineering with POSTECH. His current research interests include 3D reconstruction, scene understanding, image manipulation, and computer graphics.

Yeongyu Jung received his master's degree in computer science and engineering from Pohang University of Science and Technology (POSTECH), Pohang, South Korea, in 2016. He is currently working in KOG, Korea. His current research interest is rendering of computer graphics.

Haejoon Kim received his B.S. degree from Pohang University of Science and Technology (POSTECH), Pohang, South Korea, in 2014. He is currently a Ph.D. student in computer science and engineering with POSTECH. His current research interests include 3D reconstruction, semantic segmentation and computer graphics.

Seungyong Lee received his Ph.D. degree in computer science from KAIST in 1995. From 1995 to 1996, he worked at the City College of New York as a post-doc. Since 1996, he has been a professor at POSTECH, where he leads the Computer Graphics Group. During his sabbaticals, he worked at MPI Informatik (2003–2004) and Creative Technologies Lab at Adobe Systems (2010–2011). His technologies on image deblurring and photo upright adjustment have been transferred to Adobe Creative Cloud and Adobe Photoshop Lightroom. His current research interests include image and video processing, non-photorealistic rendering, and 3D scene reconstruction.