TextureFusion: High-Quality Texture Acquisition for Real-Time RGB-D Scanning

Joo Ho Lee*[1] Hyunho Ha[1] Yue Dong[2] Xin Tong[2] Min H. Kim[1]

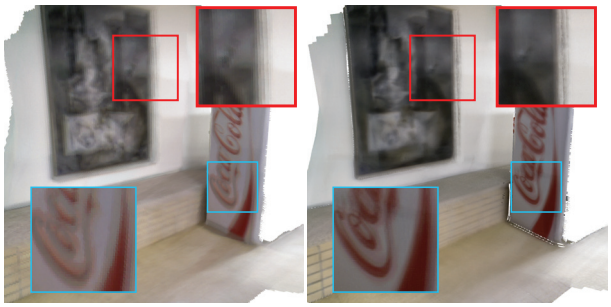
[1]KAIST [2]Microsoft Research Asia

Abstract

Real-time RGB-D scanning technique has become widely used to progressively scan objects with a hand-held sensor. Existing online methods restore color information per voxel, and thus their quality is often limited by the trade-off between spatial resolution and time performance. Also, such methods often suffer from blurred artifacts in the captured texture. Traditional offline texture mapping methods with non-rigid warping assume that the reconstructed geometry and all input views are obtained in advance, and the optimization takes a long time to compute mesh parameterization and warp parameters, which prevents them from being used in real-time applications. In this work, we propose a progressive texture-fusion *method specially designed for real-time RGB-D scanning. To this end, we first devise a novel* texture-tile voxel grid*, where texture tiles are embedded in the voxel grid of the signed distance function, allowing for high-resolution texture mapping on the low-resolution geometry volume. Instead of using expensive mesh parameterization, we associate vertices of implicit geometry directly with texture coordinates. Second, we introduce* real-time texture warping *that applies a spatially-varying perspective mapping to input images so that texture warping efficiently mitigates the mismatch between the intermediate geometry and the current input view. It allows us to enhance the quality of texture over time while updating the geometry in real-time. The results demonstrate that the quality of our real-time texture mapping is highly competitive to that of exhaustive offline texture warping methods. Our method is also capable of being integrated into existing RGB-D scanning frameworks.*

1. Introduction

Real-time RGB-D 3D scanning has become widely used to progressively scan objects or scenes with a hand-held RGB-D camera, such as Kinect. The depth stream from the camera is used to determine the camera pose and is accumulated to a voxel grid that contains surface distance. The truncated depth values in the grid become a surface representation through a marching cube algorithm [24]. The color stream, in general, is accumulated to the voxel to scan color representation in a similar manner that captures depth. However, the current color representation in real-time RGB-D scanning remains suboptimal due to the following challenges: first, since the color information is stored in each voxel in the grid, there is an inevitable tradeoff between spatial resolution and time performance [6, 13, 25, 24]. For instance, when we decrease the spatial resolution of the grid



(a) Voxel representation (b) Texture representation without optimization

(c) Global optimization only (d) Spatially-varying perspective optimization

Figure 1: We compare per-voxel color representation (a) with conventional texture representation without optimization (b). Compared with global optimization only (c), our texture fusion (d) can achieve high-quality color texture in real-time RGB-D scanning. Refer to the supplemental video for a real-time demo.

*Part of work was done during an internship in MSRA.

for fast performance, we need to scarify the spatial resolution of color information. See Figure 1(a) for an example. Second, the imperfection of the RGB-D camera introduces several artifacts: depth noise, distortion in both depth and color images, and asynchronization between the depth and color frames [31, 3, 10, 14, 9]. They lead to inaccurate estimation of imperfect geometry, color camera pose, and the mismatch between the geometry and color images (Figures 1a and 1b).

These challenges could be mitigated by applying a texture mapping method that includes global/local warping of texture to geometry. The traditional texture mapping methods assume that the reconstructed geometry and all input views are known for mesh parameterization that builds a texture atlas [18, 27, 30]. Moreover, the existing texture optimization methods calculate local texture warping to register texture and geometry accurately [10, 14, 9, 31, 3]. However, these methods also require long computational time for optimization and are not suitable for real-time RGB-D scanning. For example, a state-of-the-art texture optimization method [31] takes five to six minutes to calculate non-rigid warp parameters of 30 images for a given 3D model. It is inapplicable for real-time 3D imaging applications.

In this work, we propose a progressive texture fusion method, specially designed for real-time RGB-D scanning. To this end, we first develop a novel texture-tile voxel grid, where texture tiles are embedded in the structure of the signed distance function. We integrate input color views as warped texture tiles into the texture-tile voxel grid. Doing so allows us to establish and update the relationship between geometry and texture information in real-time without computing a high-resolution texture atlas. Instead of using expensive mesh parameterization, we associate vertices of implicit geometry directly with texture coordinates. Second, we introduce an efficient texture warping method that applies the spatially-varying perspective mapping of each camera view to the current geometry. This mitigates the mismatch between geometry and texture effectively, achieving a good tradeoff between quality and performance in texture mapping. Our local perspective warping allows us to register each color frame to the canonical texture space precisely and seamlessly so that it can enhance the quality of texture over time (Figures 1d). The quality of our real-time texture fusion is highly competitive to exhaustive offline texture warping methods. In addition, the proposed method can be easily adapted to existing RGB-D scanning frameworks. All codes and demo are published online to ensure reproducibility (`https://github.com/KAIST-VCLAB/texturefusion.git`).

2. Related Work

In this section, we provide a brief overview of color reconstruction methods in RGB-D scanning.

2.1. Online Color Per Voxel

Existing real-time RGB-D scanning methods [24, 23, 6, 11, 12] reconstruct surface geometry by accumulating the depth stream into a voxel grid of the truncated surface distance function (TSDF) [5], where the camera pose for each frame is estimated by the iterative closest point (ICP) method [26] simultaneously. To obtain the color information together, existing methods inherit the data structure of TSDF, restoring a color vector to each voxel in the volumetric grid. While this approach can be integrated easily into the existing real-time geometry reconstruction workflow, the quality of color information is often limited by the long-lasting tradeoff between spatial resolution and time performance. In addition, with the objective of real-time performance, neither global nor local registration of texture has been considered in real-time RGB-D scanning. Due to the imperfectness of the RGB-D camera, these real-time methods often suffer from blurred artifacts in the captured color information. In contrast, we perform real-time texture optimization to register texture to geometry progressively. To the best of our knowledge, our method is the first work that enables the non-rigid texture optimization for real-time RGB-D scanning.

2.2. Offline Texture Optimization

The mismatch between texture and geometry is a long-lasting problem, which has triggered a multitude of studies. First, to avoid the mismatch problem of multi-view input images, an image segmentation approach was introduced to divide surface geometry into small segments and map them to the best input view [17, 28, 10, 9, 19]. However, the initial segmentation introduces seam artifacts over surface segment boundaries, requiring an additional optimization step to relax the misalignment. They optimize small textures on each surface unit by accounting for photometric consistency [10, 9], or estimating a flow field from the boundaries to the inner regions on each surface unit [19]. However, the image segmentation per surface unit easily loses the low-frequency base image structure. It often requires additional global optimization over all the multi-view images after the scanning completes, making online scanning of these images infeasible. Also, this process cannot be applied to the progressive optimization of real-time RGB-D scanning. In contrast, our spatially-varying perspective warping method is capable of being applied to online scanning, allowing for seamless local warping of input images without misalignment artifacts.

Second, a warping approach was proposed to transform input images to mitigate the misalignment between geometry and texture by proposing local warp optimization. Aganj et al. [1] leverage SIFT image features to estimate a warp field. The sparse field is then interpolated through thin plate geometry. Dellepiane et al. [8] make use of hierarchical

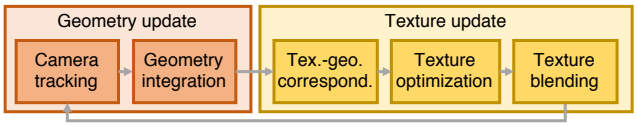


Figure 2: Overview. Our workflow consists of two main steps: Once a new depth map and color image are acquired, the camera parameters and geometry information are updated. After that, the correspondence between new geometry and texture is computed. Finally, the new image is blended to the texture space via perspective warping.

optical flow to achieve non-local registration of multi-view input. However, the former is less accurate in the interpolated region on the flow field; the later requires complex flow interpolation. Then, computational cost increases accordingly.

Recently, the global optimization of multi-view input was proposed to achieve high-quality texture mapping by not only optimizing the local mismatch of texture but also adjusting the global alignment of multi-view input. Zhou et al. [31] optimize the camera pose and apply affine-based warp fields to align them to the surface geometry. Bi et al. [3] synthesize novel intermediate views with photometric consistency by backprojecting neighboring views. Again, they also require global optimization over all the multi-view images after the scanning completes in order to estimate optical flow, making them incapable of being applied to online scanning. In contrast, we devise a spatially-varying projective warping method that is more efficient than the flow-based approaches, enabling real-time texture fusion. Our non-rigid warping field aligns per-frame color information to the texture-voxel grid progressively, resulting in high-quality texture mapping competitive to offline warping optimization.

3. Online Texture Fusion

Our online texture fusion method is enabled by two main contributions: a novel texture-tile voxel grid (Section 3.1) and an efficient, non-rigid texture optimization method (Section 3.3).

Overview. Our real-time RGB-D scanning iteratively repeats two main steps for geometry and texture to progressively accumulate both pieces of information in a canonical space. The geometric reconstruction part is based on an existing real-time 3D scanning method [25], and our texture reconstruction part consists of three main sub-steps: (1) finding the texture-geometry correspondence after updating geometry in the canonical space, (2) calculating the spatially-varying perspective warping of the current view, and (3) blending the warped view with texture in the canonical space. Figure 2 provides an overview of our workflow.

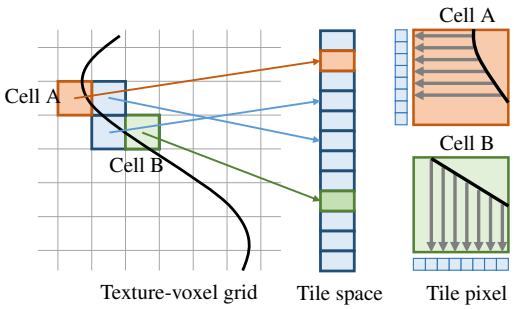


Figure 3: Our texture-tile data structure is integrated to the grid of the SDF. For every zero-crossing cell, we assign a 2D regular texture tile. To associate surface and texture, the texture tile is projected along the orthogonal axis direction.

3.1. Texture-Tile Voxel Grid

The first challenge for real-time texture fusion is that both geometry and texture need to be updated progressively every frame, and thus the existing mesh parameterization and texture atlas are not suitable for real-time scanning.

In previous studies, the tile-shaped texture architecture has been proposed to achieve the optimal compactness of a texture atlas, often using a tree structure for rendering with texture [16, 7, 2]. One benefit is that no mesh parameterization is required. With the objective of real-time scanning, we are inspired to combine the concept of texture tiles with the voxel grid of the signed distance function (SDF). By doing so, the relationship between an implicit surface and a texture tile within a voxel can be maintained through orthographic projection, rather than relying on expensive meshing and mesh parameterization. This key idea allows us to update the texture information progressively in real-time scanning.

We introduce a novel *texture-tile voxel grid* that consists of a regular lattice grid in 3D, where each grid point has an SDF value and the index of a texture tile. A cell, a basic unit of surface reconstruction, includes eight adjacent grid points. An implicit surface in the cell is defined by connecting a set of zero-crossing points through the trilinear interpolation of SDF values. We assign a fixed size of texture patches, so-called *texture tiles*, to zero-crossing cells. A pixel of texture tiles contains three properties: a weight, color and depth. We update these properties of each tile dynamically over time.

The mapping between texture and geometry is performed by orthogonal projection. To minimize the sampling loss, we select one of the axis directions, where the projection area is the maximum. We assume that the size of a cell is small so that only, at most, only one facet exists and is associated with one texture tile in a cell. This holds because our texture cell shares the same resolution as the SDF that contains the reconstructed geometry. Figure 3 depicts the data structure of the texture-tile grid and the orthographic projection.

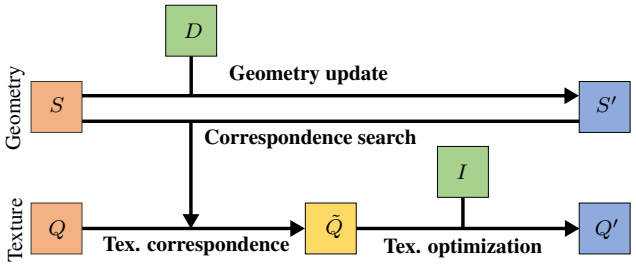
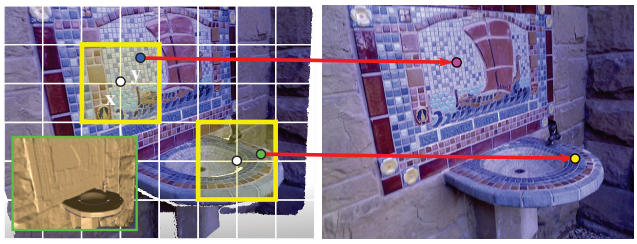


Figure 4: We first update geometry S with input depth D, yielding new geometry S'. By computing new texture correspondences, we update Q to yield the intermediate texture $\tilde{Q}$. We warps input image I via non-rigid texture optimization and then integrate it to new texture Q'.



Geometry S' and Texture $\tilde{Q}$ Image I

Figure 5: To register the current camera image I and the rendered image (using the current texture $\tilde{Q}$ and new geometry S'), we first estimate a camera motion field of pixel $\mathbf{x}$ to increase the photo-consistency of the local window (yellow box). We then integrate the warped I to the texture in the canonical space using the inverse motion field.

3.2. Texture-Geometry Correspondence

Updating both geometry and texture is a chicken-and-egg problem because the texture correspondence no longer holds anymore when either of them changes. This is also the reason why all the texture optimization methods choose a global optimization manner; however, this makes them offline. We, therefore, propose a workflow to update both pieces of information progressively. See Figure 4 for an overview.

Suppose we estimate the current camera pose $\mathbf{C}$ through ICP using the current depth values D. We first update surface geometry from S to S' by integrating the current depth values D in the texture-tile voxel grid. In consequence, the correspondence between the current texture Q and new geometry S' is no longer valid. There is no ground truth nor any certain prior in searching for the new correspondences. Therefore, we find out new texture correspondences by exhaustively searching corresponding points on previous geometry S (associated with texture Q) from the new surface geometry S' along its normal directions, yielding a new intermediate texture space $\tilde{Q}$ that corresponds to S'.

3.3. Efficient Texture Optimization

The second challenge for high-quality texture fusion is that the *perfect* geometric correspondence between geometry and multi-view input does not hold in real RGB-D scanning. First, the depth and color camera modules in conventional RGB-D cameras, such as Kinect, PrimeSense, and Xtion, are not fully synchronized, i.e., the depth and color frames are captured at different times [31, 3]. The global non-synchronization of color and depth can be easily fixed by a global perspective warp. Second, the rolling shutter artifacts often occur, and input frames are spatially warped [15]. This means that asynchronous time offset could exist in different regions; therefore, a local perspective warp can mitigate this problem. Lastly, some geometry errors in 3D scanning, especially with low frequency, can be fixed by a perspective warp. Of course, all those mismatch problems can be fixed by a free-form warp. We realized that these solutions essentially become *perspective warp*, and, thus, we devise a novel, efficient texture optimization based on spatially-varying perspective warping.

Spatially-varying perspective warp. To address local errors after the global registration, many free-form warping, such as optical flow or affine transformation, have been proposed [8, 31, 3]. Patch-based optical flow is powerful but relatively slow, and the affine transformation often introduces local errors. There is an inevitable tradeoff between quality and performance in local texture optimization. We therefore propose an alternative approach of *2D perspective warp*, estimating the camera motions of local perspective windows, inspired by the moving direct linear transformation (DLT) [29].

The camera pose $\mathbf{C}$ allows us to register the current camera image to the canonical space of the texture-tile voxel grid in a certain direction. When the color camera captures a part of the scene from the camera pose $\mathbf{C}$, the partial geometry information S' can be presented as a point cloud $\tilde{P}$. The current camera view of the new geometry can be expressed as $I(\pi(\mathbf{C}\tilde{P}))$, where $\pi()$ is the perspective projection function with known camera intrinsics. See Figure 5.

To address local warping, we assume that the current color frame (captured by the RGB camera) consists of local windows ν (the yellow boxes about pixel $\mathbf{x}$ on the 2D grid in Figure 5) and that each local window $\nu(\mathbf{x})$ is captured by each virtual sub-camera with minor different pose $\mathbf{T}(\mathbf{x})$. We approximate the minor camera motion in 3D as $\mathbf{T}(\mathbf{x})$ in the form of linearized 6-DOF transformation using twist representation [22]. This minor transformation $\mathbf{T}(\mathbf{x})$ can be multiplied to $\mathbf{C}$ to account for the local motion of sub-windows.

Our main objective is to register the current camera view to the texture information with respect to the newly updated geometry in the canonical space. We first render[1] an im-

[1]For every pixel, we cast a ray to find an intersection point with the newly updated implicit surface S'. We find a corresponding cell that con-

age $\tilde{I}$ at current camera pose $\mathbf{C}$ using the newly updated geometry S' with the intermediate texture $\tilde{Q}$. Now we want to enforce the photometric consistency of the current image to the texture map through the local window ν by formulating the following objective function:

$$E(\mathbf{T}(\mathbf{x})) = \sum_{\mathbf{y}\in\nu(\mathbf{x})} \left(w\left(\mathbf{y}\right) \left(I\left(\pi\left(\mathbf{T}(\mathbf{x})\mathbf{C}\tilde{P}\left(\mathbf{y}\right)\right)\right) - \tilde{I}\left(\mathbf{y}\right)\right)\right)^2, \quad (1)$$

where $\mathbf{y}$ is a neighbor pixel within the local window $\nu(\mathbf{x})$ (see Figure 5), $w(\mathbf{y})$ is a Gaussian weight of the distance as $\exp\left(-\left\|\mathbf{y}-\mathbf{x}\right\|^2/\sigma^2\right)$ with a spatial coefficient σ. The window size ν and spatial coefficient σ controls the regularity of the estimated camera motion. If the spatial coefficient is higher and the window size becomes larger, the estimated camera motion reduces gradually to global camera motion.

To optimize a camera motion field from global camera motion to local camera motion seamlessly, we compute them in a hierarchical manner. We build multi-scale pyramids of I, $\tilde{I}$, and $\tilde{P}$ and then estimate a motion field from the coarse to the fine level. To initialize the next level, we conduct bilinear sampling from the current level of the estimated motion field. For the online performance, instead of estimating camera motions for every pixel, we estimate camera motions $\mathbf{T}(\mathbf{x})$ of points of a regular lattice grid at each level and then interpolate them for every pixel. We empirically found that three levels are sufficient and that the width between grid points is set as 64 pixels.

Different from the original moving DLT, we estimate local camera motion as a twist matrix (with six unknown parameters), later converted to $\mathbb{SE}(3)$, instead of estimating the homography matrix (with eight unknown parameters), because we already know the 3D coordinates of the corresponding geometry. In terms of optimization, it is beneficial to reduce the number of unknowns by two. In addition, our perspective warp can provide more accurate local warping with more efficient computation. Particularly, it is more powerful when the surface geometry of the scene consists of various surface orientations, thanks to the benefits of perspective warp (see Figure 1 for an example).

3.4. Online Texture Blending

Once we know the texture correspondences and the warp transformation for the input image with respect to the latest surface geometry, we are ready to blend the image into the current texture in the canonical space. For each texel, we evaluate each image pixel by spatial resolution and registration certainty.

Resolution sensitivity. We estimate the projected areas of correspondent image pixels and compute blending weights, following Buehler et al. [4]. The size of the projected area is related to the depth and the angle between the view direction and the normal direction, i.e., the projected area is inversely proportional to depth squared and proportional to the cosine value of the view angle. We compute the area weight factor, based on the factor ρ with the minimum threshold γ_{area}:

$$w_{\text{area}}\left(\mathbf{p}\right) = \max(e^{-((1-\rho)/\sigma_{\text{area}})^2}, \gamma_{\text{area}}), \quad (2)$$

where $\rho = \left(\frac{z_{\min}}{z}\right)^2 \mathbf{n}\cdot\frac{\mathbf{p}-\mathbf{c}}{|\mathbf{p}-\mathbf{c}|}$. Here $z_{\min}$ is the minimum depth of scanning, $\mathbf{p}$ is a 3D point corresponding to each texel, $\mathbf{n}$ is the surface normal of $\mathbf{p}$, and $\mathbf{c}$ is the camera center.

Registration certainty. The registration certainty between the image and the geometry is crucial w.r.t. the texture accuracy. The texture inaccuracy, caused by the imperfect geometry, is crucial for two types of surface areas: skewed surface and near-occlusion surface. When the angle between the view direction and the surface normal becomes large, the displacement error greatly increases. Also, near the occlusion edges, the inaccurate object boundaries of objects causes severe artifacts, such as texture bleeding to forward/backward objects. To attenuate these artifacts, we first estimate occlusion edges and evaluate the soft-occlusion weight $w_{\text{occ}}(\mathbf{p})$ at point $\mathbf{p}$ that corresponds to the camera pixel $\mathbf{x} = \pi(\mathbf{C}\mathbf{p})$, by eroding occlusion edges with a discrete box kernel and a Gaussian kernel. We evaluate the angle weight factor as:

$$w_{\text{angle}}\left(\mathbf{p}\right) = \max(e^{-\left(\left(1-\mathbf{n}\cdot\frac{\mathbf{p}-\mathbf{c}}{|\mathbf{p}-\mathbf{c}|}\right)/\sigma_{\text{angle}}\right)^2}, \gamma_{\text{angle}}). \quad (3)$$

Finally, our total blending weight for a given camera is computed as

$$w\left(\mathbf{p}\right) = w_{\text{area}}\left(\mathbf{p}\right)\cdot w_{\text{occ}}\left(\mathbf{p}\right)\cdot w_{\text{angle}}\left(\mathbf{p}\right). \quad (4)$$

We update weights and colors of texture Q in an integrated manner. The previous texture $\tilde{Q}$ and the image I are blended as

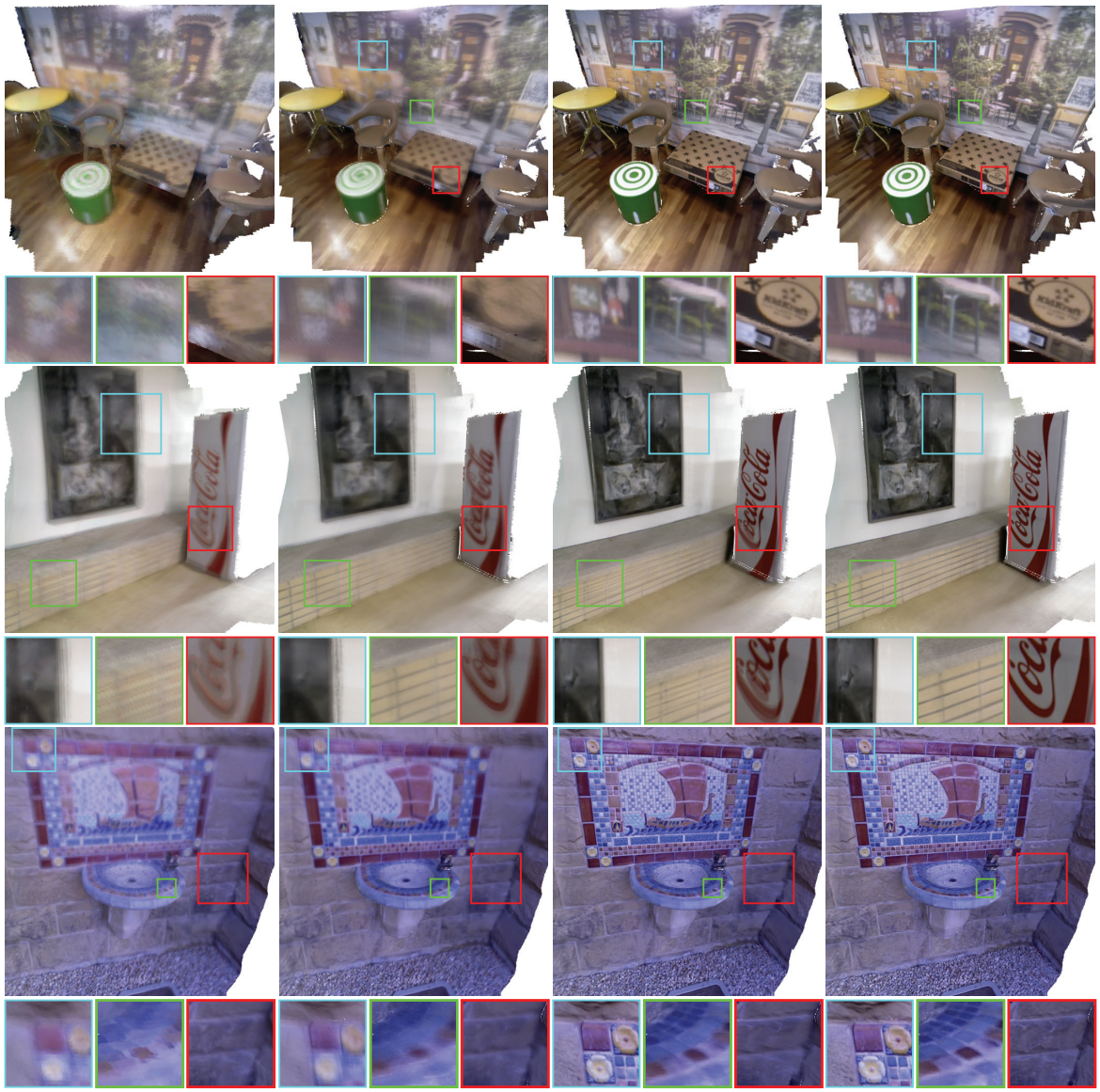
$$\begin{aligned} Q(\mathbf{p}) &= \frac{W(\mathbf{p})\tilde{Q}(\mathbf{p}) + w(\mathbf{p})I\left(\mathbf{x}'\right)}{W(\mathbf{p})+w(\mathbf{p})}, \\ W(\mathbf{p}) &= \min(W(\mathbf{p}) + w(\mathbf{p}), \psi), \end{aligned} \quad (5)$$

where ψ is a weight upper bound.

4. Results

Experimental Setup. We implemented our method by integrating our texture fusion algorithm on an existing real-time 3D scanning method [25]. We used a desktop machine equipped with Intel Core i7-7700K 4.20 GHz and a graphics card of NVIDIA Titan V. We used a commercial RGB-D camera, an Asus Xtion Pro Live, which provides imperfect synchronization of color and depth frames in the VGA resolution (640×480) in 30 Hz.

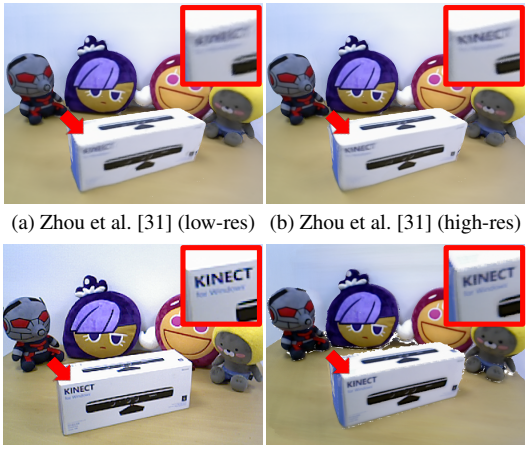
tains the intersection point and sample color in the texture tile using bilinear interpolation.



(a) Nießner et al. [25] (per-voxel representation) (b) Maillot et al. [21] (texture but no texture optimization) (c) Zhou et al. [31] (global perspective + affine warp) (d) Ours (spatially-varying perspective warp)

Figure 6: We compare our online method (d) with three existing RGB-D scanning methods: (a) the online voxel representation method [25], (b) the offline traditional texture mapping method [21], and (c) the offline texture optimization method [31]. Our method achieves high-quality texture representation without sacrificing scanning performance thanks to our spatially-varying perspective warping and the texture-tile voxel grid. Two scenes from the top are our captures, where 8×8 texture tile per $10mm$ voxel is used. The last reference scene is obtained from [31], where a 4×4 texture tile per $4mm$ voxel is used.

Comparison with Other RGB-D Scanning We compare the texture quality of our method with three other methods. See Figure 6. The first column shows scanning results by an online RGB-D scanning method [25] that uses per-voxel color representation in their implementation. The voxel-based representation in the online method suffers from the tradeoff between quality and performance. The second column presents results by a traditional offline texture mapping method [21], where no texture optimization is conducted. Owing to the correspondence mismatch between geometry and texture, the image structures are blurred severely. The third column shows an offline texture optimization method [31] that includes global warping of the color camera motion and local affine transformation. We experimented with our implementation of [31] on a GPU for this experiment. The results of the offline texture optimization still present misalignments in local regions despite long computation time. The last column shows the results of our online texture optimization method. Thanks to the efficient spatially-varying perspective warp, we can achieve both high-quality texture and real-time performance.

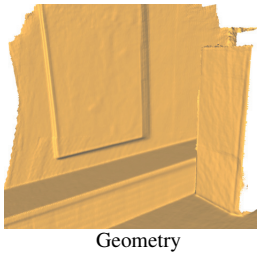


(a) Zhou et al. [31] (low-res) (b) Zhou et al. [31] (high-res)

(c) Scene image (d) Ours

Figure 7: Comparison of our online texture optimization against an offline texture optimization method [31].

Implementation details. For the backward compatibility of our method, we implement a conversion from our texture model to a conventional texture atlas, associated with an ordinary surface model. We first create mesh representation by the marching cube algorithm [20] and then create a lattice-shaped texture atlas by finding out zero-crossing cells in the SDF to pack texture tiles of each cell into the texture atlas. By computing the local 3D coordinates within the cell and projecting them in the selected axis direction, we compute local 2D texture coordinates of the vertices of facets. The local texture coordinates converted to the global texture coordinates associated with the generated texture map. Note that the origin of the local texture coordinates is determined by the new tile index. To avoid texture bleeding on the boundary among discrete tiles, we add one-pixel padding for every tile. See the inset figure on the right for an example of a converted 3D model and its texture atlas (the scene model shown in Figure 1).



Geometry

Texture map

Comparison with other offline optimization. We compared our online method with an offline texture optimization method [31]. For this experiment, we used the authors' implementation [32]. See Figure 7. For the fairness of comparison, we share the same geometry model. Our method used the voxel size of 4 mm in each dimension with a 4×4 texture tile in each cell, and since Zhou's method calculates the final output color per vertex, we increased the spatial resolution of the 3D model, where each vertex distance is around 1 – 2 mm. Even though Zhou's method has two to four-times higher spatial resolution than ours, the spatial resolution of texture information of our method is significantly higher, given the same input images.

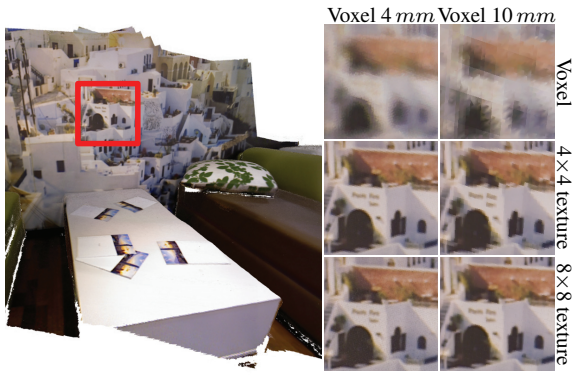


Figure 8: Impacts of resolution parameters. Our method can achieve high-resolution texture even with a low-resolution voxel grid. The texture quality of our method is significantly higher than that of existing voxel representation.

Texture tile, voxel unit	Geometry integration	Texture correspondence	Texture optimization	Texture blending	Total time per frame
4×4 tile, 4 mm	3.7 ms	4.1 ms	30.5 ms	2.3 ms	40.6 ms
8×8 tile, 4 mm	3.8 ms	4.0 ms	40.6 ms	2.4 ms	50.7 ms
4×4 tile, 10 mm	3.7 ms	2.3 ms	25.8 ms	1.4 ms	33.3 ms
8×8 tile, 10 mm	3.7 ms	2.3 ms	27.1 ms	1.5 ms	34.6 ms

Table 1: Per-function performance measures of our method with different resolutions. We mainly use the first row configuration of a 4×4 tile and 4 mm voxel size.

Impacts of resolution. Figure 8 shows the impact of the resolution parameter (the resolutions of the voxel grid and the texture tile) in our method on the texture quality. In our method, the configuration of 8×8 and 4 mm does not increase texture quality any further. We empirically choose the configuration of 4×4 tile and 4 mm voxel size. Table 1 provides averaged per-function performance measures of our method with different resolutions. Even including texture optimization for every frame, we achieve 25 frames-per-second (fps) in runtime. We found that texture optimization does not need to be computed every frame in general scanning, and thus we computed the optimization process every fourth frame so that we achieved 25–35 fps in our video results. As the texture resolution increases, it increases the rendering time so optimization time increases. Those numbers are trade-offs based on the scene, RGB-D sensor and GPU computation. Our method is not limited to a specific camera, VGA input, etc., but is flexible to extend to other devices as well.

Comparison of warping methods. Figure 9 compares our perspective warping method with different warping methods in terms of the accuracy. For fair comparison, we use hyperparameters, such as the same cell width and hierarchical levels. The local affine warping cannot correct the pose error due to the model limitation. The global perspective warping method reduces the misalignment globally, but local errors still remain. The spatially-varying perspective warping model robustly corrects local errors of registration,

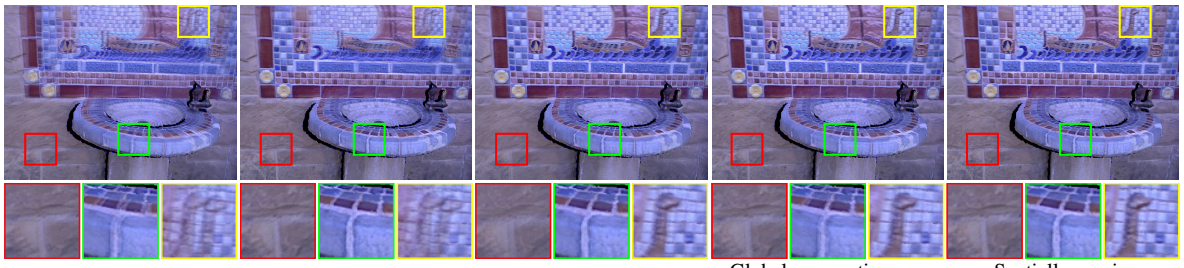


Figure 9: Comparison of different warping methods. Different warp methods are tested with the same parameters, such as the number of pyramid levels, the width of lattices, and others as possible. The affine-based warp method often fails to correct global camera motion. While even the combination of affine warp and global perspective warp still has errors, our perspective warp method reduces the ghosting artifact significantly.

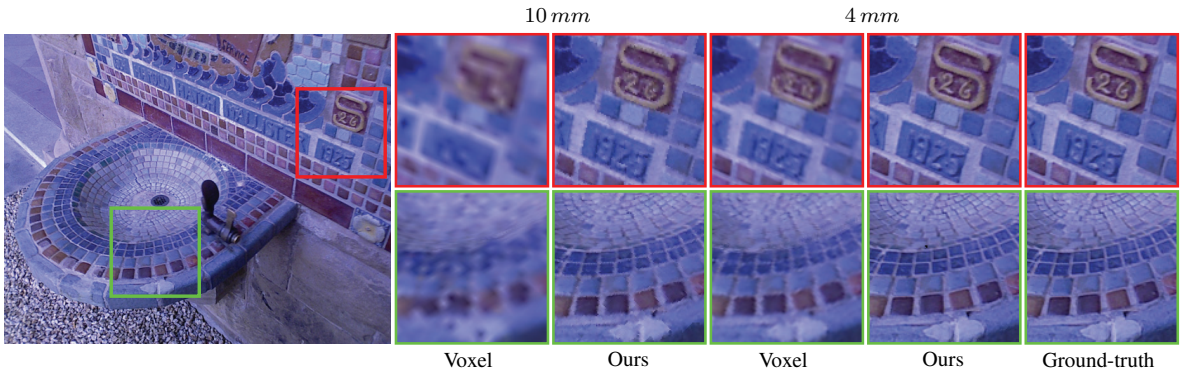


Figure 10: Comparison of quantization errors between volumetric representation and our textile representation (4×4 texture tile). We unproject an image on each representation and render it to compare quantization errors. In our experiment, 4×4 patch size with a 4 mm voxel size is sufficient in terms of image quality.

while the combination of global perspective warp and local affine warp cannot correct perfectly.

Comparison of quantization error. Figure 10 compares quantization errors with different configurations. We back-project an image on the texture space and then render the texture in the same image space. For the VGA image resolution, a 4×4 texture tile in a 4 mm voxel unit is sufficient to project high-frequency information on the texture space.

5. Discussion

The proposed method is not free from limitations, which can lead to interesting future work. Our method evaluates the photometric consistency of the texture and input images. View-dependent appearance or severe exposure change might degrade the quality of texture reconstruction. Particularly, the new geometry scanning with blunt change of exposure could cause a problem in the texture-to-image registration step. In addition, we just capture objects' color directly as texture. There is no separation process of incident color to diffuse color and scene illumination. Capturing diffuse color as texture is interesting future work.

We found that our method is particularly effective when the captured scene is large and includes various surface orientations. Capturing even larger scale scenes would be worthy to explore as future work.

6. Conclusions

We have presented a high-quality texture acquisition method for real-time RGB-D scanning. We have proposed two main contributions: First, our novel texture-tile voxel grid combines the texture space and the signed distance function together, allowing for high-resolution texture mapping on the low-resolution geometry volume. Second, our spatially-varying perspective mapping efficiently mitigates the mismatch between geometry and texture. While updating the geometry in real-time, it allows us to enhance the quality of texture over time. Our results demonstrated that the quality of our real-time texture mapping is highly competitive to that of existing offline texture warping methods. We anticipate that our method could be used broadly as our method is capable of being integrated into existing RGB-D scanning frameworks.

Acknowledgements

Min H. Kim acknowledges Korea NRF grants (2019R1A2C3007229, 2013M3A6A6073718), Samsung Research, KOCCA in MCST of Korea, and Cross-Ministry Giga KOREA (GK17P0200).

References

[1] E. Aganj, P. Monasse, and R. Keriven. Multi-view texturing of imprecise mesh. Mar. 2009.

[2] D. Benson and J. Davis. Octree textures. *ACM Trans. Gr.*, 21(3):785–790, July 2002.

[3] S. Bi, N. K. Kalantari, and R. Ramamoorthi. Patch-based optimization for image-based texture mapping. *ACM Trans. Graph.*, 36(4), July 2017.

[4] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 425–432, 2001.

[5] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, 1996.

[6] A. Dai, M. Niessner, M. Zollhofer, S. Izadi, and C. Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Trans. Graph.*, 36(4), May 2017.

[7] Da. (g.) DeBry, J. Gibbs, D. D. Petty, and N. Robins. Painting and rendering textures on unparameterized models. *ACM Trans. Gr.*, 21(3):763–768, July 2002.

[8] M. Dellepiane, R. Marroquim, M. Callieri, P. Cignoni, and R. Scopigno. Flow-based local optimization for image-to-geometry projection. *IEEE Transactions on Visualization and Computer Graphics*, 18(3), Mar. 2012.

[9] Y. Fu, Q. Yan, L. Yang, J. Liao, and C. Xiao. Texture mapping for 3d reconstruction with rgb-d sensor. In *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, June 2018.

[10] R. Gal, Y. Wexler, E. Ofek, H. Hoppe, and D. Cohen-Or. Seamless montage for texturing models. *Computer Graphics Forum*, 29(2):479–486, 2010.

[11] K. Guo, F. Xu, T. Yu, X. Liu, Q. Dai, and Y. Liu. Real-time geometry, albedo, and motion reconstruction using a single rgb-d camera. *ACM Trans. Graph.*, 36(3), June 2017.

[12] K. Guo, F. Xu, T. Yu, X. Liu, Q. Dai, and Y. Liu. Real-time high-accuracy 3d reconstruction with consumer rgb-d cameras. *ACM Trans. Graph.*, 1(1), May 2018.

[13] M. Innmann, M. Zollhofer, M. Niessner, C. Theobalt, and M. Stamminger. Volumedeform: Real-time volumetric non-rigid reconstruction. In *Proc. Euro. Conf. Comput. Vis.*, Mar. 2016.

[14] J. Jeon, Y. Jung, H. Kim, and S. Lee. Texture map generation for 3d reconstructed scenes. *The Visual Computer*, 32(6-8):955–965, 2016.

[15] C. Kerl, J. Stückler, and D. Cremers. Dense continuous-time tracking and mapping with rolling shutter rgb-d cameras. In *Proc. IEEE Int. Conf. Comput. Vis.*, pages 2264–2272, 2015.

[16] S. Lefebvre and C. Dachsbacher. Tiletrees. In *ACM SIGGRAPH Symp. Interactive 3D Graph. Games.*, May 2007.

[17] V. Lempitsky and D. Ivanov. Seamless mosaicing of image-based texture maps. In *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, pages 1–6, June 2007.

[18] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *ACM transactions on graphics (TOG)*, volume 21, pages 362–371. ACM, 2002.

[19] W. Li, H. Gong, and R. Yang. Fast texture mapping adjustment via local/global optimization. *IEEE Trans. Visualization. Comput. Graph.*, pages 1–1, 2018.

[20] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4), Aug. 1987.

[21] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 27–34. ACM, 1993.

[22] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994.

[23] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, June 2015.

[24] 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 *IEEE Int. Symp. Mixed and Augmented Reality*, Oct. 2011.

[25] M. Niessner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6):169:1–169:11, Nov. 2013.

[26] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, May 2001.

[27] Pedro V Sander, Steven Gortler, John Snyder, and Hugues Hoppe. Signal-specialized parameterization. 2002.

[28] M. Waechter, N. Moehrle, and M. Goesele. Let there be color! large-scale texturing of 3d reconstructions. In *Proc. Euro. Conf. Comput. Vis.*, pages 836–850, 2014.

[29] J. Zaragoza, T. Chin, M. S. Brown, and D. Suter. As-projective-as-possible image stitching with moving DLT. In *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, pages 2339–2346, 2013.

[30] Kun Zhou, John Synder, Baining Guo, and Heung-Yeung Shum. Iso-charts: stretch-driven mesh parameterization using spectral analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 45–54. ACM, 2004.

[31] Q. Zhou and V. Koltun. Color map optimization for 3d reconstruction with consumer depth cameras. *ACM Trans. Graph.*, 33(4), July 2014.

[32] Q. Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.