This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2831220, IEEE Transactions on Visualization and Computer Graphics

Fast Texture Mapping Adjustment via Local/Global Optimization

Wei Li, Huajun Gong, Ruigang Yang *Senior Member, IEEE*

Abstract

This paper deals with the texture mapping of a triangular mesh model given a set of calibrated images. Different from the traditional approach of applying projective texture mapping with model parameterizations, we develop an image-space texture optimization scheme that aims to reduce visible seams or misalignment at texture or depth boundaries. Our novel scheme starts with an efficient local (and parallel) texture adjustment scheme at these boundaries, followed by a global correction step to rectify potential texture distortions caused by the local movement. Our phased optimization scheme achieves 50∼100 times speed-up on GPU (or 6× on CPU) compared to previous state-of-the-art methods. Experiments on a variety of models showed that we achieve this significant speed-up without sacrificing texture quality. Our approach significantly improves resilience to modeling and calibration errors, thereby allowing fast and fully automatic creation of textured models using commodity depth sensors by untrained users.

Index Terms

Texture mapping, Parameterization, Texture optimization

- *Wei Li is with the Department of Automatic Control, Nanjing University of Aeronautics and Astronautisc, Nanjing, China; the University of Kentucky, Lexington, Kentucky, USA; the Baidu Research, Beijing, China; and National Engineering Laboratory of Deep Learning Technology and Application, China. E-mail: liweimcc@gmail.com*
- *Huajun Gong is with the Department of Automatic Control, Nanjing University of Aeronautics and Astronautisc, Nanjing, China. E-mail: ghj301@nuaa.edu.cn*
- *Ruigang Yang is with the University of Kentucky, Lexington, Kentucky, USA; the Baidu Research, Beijing, China; National Engineering Laboratory of Deep Learning Technology and Application, China. E-mail: ryang@cs.uky.edu*

Manuscript received ...

Fast Texture Mapping Adjustment via Local/Global Optimization

1 Introduction

With the wide availability of low cost commodity 3D full frame sensors, such as the Microsoft Kinect and Intel Real-Sense, it is becoming easier and easier to capture 3D data. Starting from the robust system of KinectFusion [1], [2], large scale scanning [3], [4], to the more recent work of fusion of dynamic objects over space and time [5], [6], good quality 3D models can be obtained even with a single hand-held depth camera. These approaches typically use a volumetric representation of 3D geometry, the final model is mostly presented as a white (texture-less) model or colored with simple per-vertex color. Given the high cost associated with a dense voxel grid, per-vertex colored model appears to be blurry and low resolution, the final model's appearance captured by these methods left something to be desired.

Ideally, the 3D model should be textured by color images. Many depth sensors come with a color texture camera. In addition, color camera's resolution is at least a few times higher than the depth sensor's resolution. However, directly projecting the images onto the model is problematic. Due to a number of factors, including model inaccuracy, calibration error, and camera exposure variations, misaligned patches and color seams are often visible in the textured model.

Rather than direct projecting, we present a novel framework that *optimizes* both the texture coordinates and texture color for each triangle on the model, converting a 3D model with a set of calibrated images to a seamlessly textured 3D model. Our framework has the following advantages: 1) it is robust to geometric inaccuracy in the model and image-model registration error; 2) it is memory efficient and scalable, the optimization is carried out sequentially in the input image space. There is no need to parameterize the 3D model, and new images can be easily added. This property makes it ideal for scanning large areas since there is no global texture atlas to be built; and most importantly 3) it is fast, over dozens times on GPU (or six times on CPU) than the previous state of the art methods [7] without sacrificing any quality.

We are certainly not the first group to realize the importance of color or texture for 3D models. Most related to our formulation, Lempitsky et al. [8] developed a global optimization scheme that favors smoothness in the texture assignment and penalizes the introduction of sharp seams. Gal et al. [7] further advanced the state-of-the art by introducing small shift of texture coordinates to achieve the most robust texture mapping results against mis-registration and model inaccuracy, however, the additional adjustment is computationally expensive. When dealing with large data sets, Waechter el al. [9] abstained from this approach since it "becomes infeasible for realistic dataset sizes". Our approach solved this computational complexity problem by using a phased and parallelizable optimization scheme.

Fig. 1 shows texturing a scanned human model. Unlike rigid object, human subjects are prone to slight movement during scanning, in particular extended arms. Thus, non-rigid point cloud alignment was applied to create a seamless geometric model, which caused mis-registration with respect to the texture cameras. With only view selection and texture blending, artifacts are visible (Fig. 1(c)). With the additional texture coordinate adjustment, our method generates better result in Fig. 1 (b).

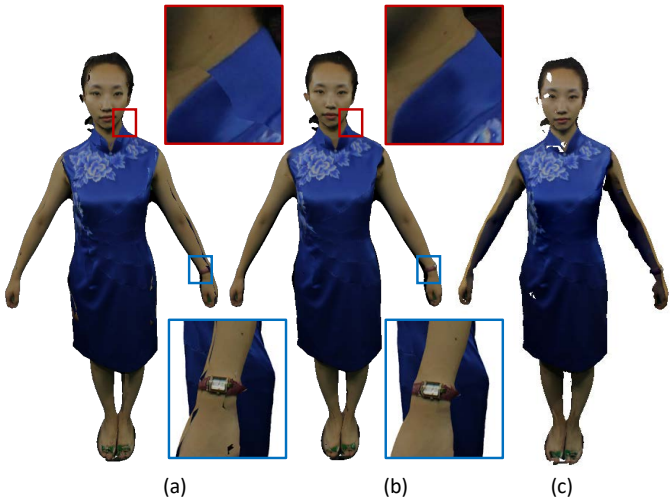


Fig. 1. (a) Textured human model using projective texture mapping,the model had small movement during the scanning process. (b) result of our method, (c) result of a recent texturing technique [9], untextured triangles are removed by design. Notice the arms where there were biggest movements.

In summary, the primary technical contribution of this paper is a flexible and computationally effective texture optimization scheme for a 3D mesh model with registered images. Given the ease of geometry acquisition with these low cost sensors, we hope our method can significantly improve the model quality, in particular its appearance, to a level that can be appreciated by average users, making 3D modeling as ubiquitous as taking pictures.

2 Related Work

Our focus is on texturing the reconstructed surface model with calibrated input images. The baseline method is to project the surface to each view to retrieve color. This method works fine if both the calibration and surface geometry are accurate. However, this is difficult to achieve with low cost scanners such as the Kinect sensor. To mitigate the visual artifacts of misaligned texture seams, blending of several images near the seam can be applied. Methods in this category are different in how the blending weights are generated. Wang et al. [10] optimize the weighting based on sampling theory, Baumberg et al. [11] using a multi-band method to blend low and high frequency contents differently. Then Totz et al. [12] introduce a multi-band weighting based on the viewing distance and angle. Blending can remove sharp seams efficiently with proper weight. However, it will lead to ghosting and blurring artifacts when images are misaligned in the overlap areas.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2831220, IEEE Transactions on Visualization and Computer Graphics

Different from blending, it has become popular to project the model surface onto images and formulate the texturing problem as an image stitching problem. Rocchini et al. [13] select textures by least angle, smooth out the selection along adjacent vertices to obtain larger contiguous regions that map to the same image, and only average border faces using barycentric coordinates as weights after a local linear registration. Lempitsky et al. [8] apply a global optimization based on a Markov Random Field(MRF) to favor smoothness in texture assignment and penalize sharp seams, followed by a seam leveling scheme. Dessein [14] present a method to blend sampled images by segmenting the image into patches. Waechter et al. [9] extended this line of work for large scale models, including additional constraints such as occluders.

Image stitching based methods alleviate many artifacts. However, some seams still remain due to misalignment and geometry inaccuracy. To deal with misalignment, Soler et al. [15] optimize a set of imaging transformations for texture synthesis, Eisemann et al. [16] compute such image space translation using optical flow and Aganj et al. [17] simplify dense optical flow computation by aligning sparse matched SIFT keypoints. Then, Gal et al. [7] extend Lempitsky et al.'s [8] work by appending a per-face translation in image space into the global MRF solver, which significantly improves the final texture quality. However, this method is time costly, and cannot guarantee global minimal. Our method is inspired by Gal et al.'s work, but with a very different optimization strategy. We use a local/global approach, which starts with a parallel adjustment of per-face texture coordinates, followed by a global warping stage that is formulated as a linear system.

Alternatively, Zhou et al. [18] proposed a new method that optimizes the camera extrinsic and intrinsic parameter using a vertex color consistency measurement to deal with misalignment. Based on Zhou et al.'s approach, the most recent method [19] uses patch-based texture synthesis technique to ensure patches' bidirectional similarity, which is more robust to large misalignment. While those methods can achieve significantly improved results, but they are too computational expensive. Even the baseline work [18], the typical running time is in the order of minutes.

In the meantime, alternative techniques based on user specifying a number of matching points between the triangle mesh and image region are studied [20], [21], [22], [23]. Different with such interactive method, our method automatically selects content from images.

3 Framework Overview

The problem we address in this paper is to texture a 3D model with a set of calibrated images. More specifically, we are given a mesh with m triangles $\{F_1, F_2...F_m\}$, and n input images $\{I_1, I_2...I_n\}$. We aim to calculate the texture coordinates (T_i) for each triangle F_i, which is denoted as $T_i = \{l_i, t_i\}$, where $l_i \in \{1...n\}$ identifies the source image, and $t_i \in \mathbb{R}^6$ is the UV coordinates in the selected source image space. Through the calibration matrices associated with input images, T_i can have multiple choices. We here present a framework to optimize $\{T_1, T_2...T_m\}$ so that the resulting textured model can have a visually pleasing appearance, without visible distortions or seams.

Our framework has two stages, the first is to select the source image for T_i, i.e., determining l_i. This is based on a number of criteria, such as the number of visible vertices, viewing angles, and smoothness constraints. More details will be presented in Section 4. The second stage is to optimize the texture coordinates

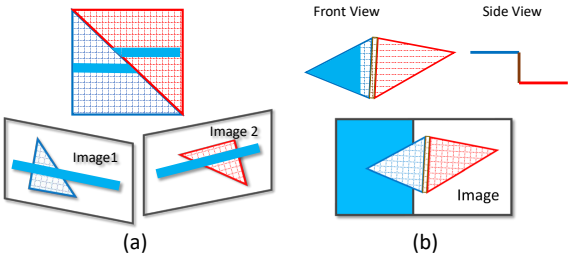


Fig. 2. Misalignments in texture mapping. (a) Misalignment between different input images, (b) misalignment near depth edges, likely due to inaccurate geometry or calibration.

t_i. A common problem in texture mapping scanned objects is the misalignment of textures. There are actually two types of misalignments. As illustrated in Fig. 2, the first is the misalignment between adjacent triangles with different input images. And the second is misalignment on depth boundaries. For example, the edge of a table has the texture of the floor or vice versa. This can happen even if there is only one input image. However, no matter what have caused the misalignments, they can always be fixed by shifting the texture coordinates of that triangle. While previous methods (e.g., [7]) in removing texture seams have been developed to solve this texture shift problem in a global optimization framework, we first recognize that the texture shift can be first optimized at the texture seams, in parallel on a per triangle basis, and then be consolidated globally over the entire image. This insight leads to a new algorithm that is over 50∼100 times faster than previous approaches, without sacrificing the texture quality. The details will be presented in Section 5.

4 Preprocessing

When a model is acquired with a continuous capture device, for example, the Kinect sensor or just a regular video camera, there could be many images that are associated with the model. Feeding all these images directly to the image labeling process is computationally expensive since each triangle may have hundreds or even thousands of potential labels to select from. Instead of just sub-sampling images using a uniform interval, we use a simple greedy method that counts the number of visible triangles in each image. If the number of *newly visible* triangles is above a certain threshold, we select that image to the source image set.

Another factor we need to consider is the model resolution. If a triangle on a model has a very small footprint on the image, for example, just a few pixels, our optimization scheme will be excessively slow and difficult to converge. Experiments have shown that good results can be obtained when a triangle edge's projection on the image is between 10 to 50 pixels long. Therefore, for dense models, we apply the edge collapse method [24] to simplify the input models to a proper density. It should be noted that this simplification process is traceable and reversible. The texture coordinates in the simplified model can be propagated to the original model.

4.1 Image Labeling

As every triangle on a mesh could be seen by more than one view from the input image set. We first select one best image source l_i for each triangle. A simple solution of such selection problem can be based on the angle between triangle surface normal and the

view direction associated with each image. This is likely to create many fragmented texture patches. We solve such a image labeling problem based on Lempitsky et al.'s [8] method. More specifically, we denote the source image for triangle F_i as I_{l_i}, where l_i is the label. The optimal label set $\mathcal{L} = \{l_1, l_2 ... l_n\}$ is obtained when the following cost function is minimized:

$$E(\mathcal{L}) = \sum_{i \in M} \left(E_{\text{data}}(l_i) + \lambda \sum_{j \in \mathcal{N}_{F_i}} E_{\text{smooth}}(l_i, l_j) \right) \quad (1)$$

In the above cost function, E_{data} is the data term, indicates the cost of assigning image I_{l_i} to F_i. We just employ two main factor in our data term: the resolution in image space for F_i and the angle between triangle normal and the view direction, it is formulated as $E_{\text{data}}(l_i) = -d(F_i, I_{l_i})a(F_i, I_{l_i})$, where $d(F_i, I_{l_i})$ denotes the mean distance between triangle F_i and the center of projection for I_{l_i}, it is an approximation of F_i's projection size on I_{l_i}. And $a(F_i, I_{l_i})$ denotes the angle between the triangle normal and the view direction. The second part of Eq. (1) is the smoothness term, where $\mathcal{N}_{F_i}$ indicates the 1-ring neighbor of triangle F_i. The pairwise smoothness cost is defined after the Potts Model $E_{\text{smooth}}(l_i, l_j) = [l_i \neq l_j]$ according to [8].

The cost function in Eq. (1) can be effectively optimized with Graph Cut [25]. Fig. 3 shows the color-coded view selection results without (b) and with (c) the spatial smoothness constraint. Actually, there are plenty of algorithm variations in choosing specific data and smoothness term [7], [8], [9]. However, our texture mapping adjusting scheme in Section 5 is open to any of those variations. After exploring several different terms, the one we choose achieves a good balance between computation and quality.

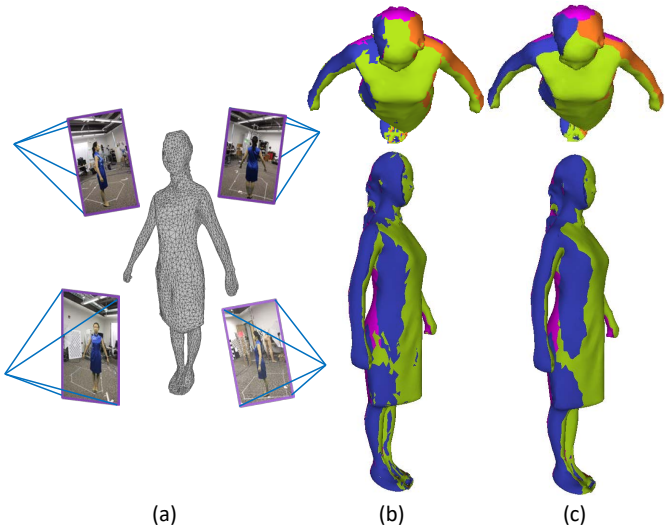


Fig. 3. (a) Four registered images for texturing human model, (b) before and (c) after apply smoothness constraint in image labeling. Different colors represent different source image labels. Texture mapping shown in (c) reduces small/isolated areas with distinctive image labels.

5 Image-space Optimization

After the image labeling, a triangle's texture coordinates can be obtained by projecting its vertices onto the selected source image. This simple approach, as we mentioned before, is likely to lead to visible texture misalignment (seams). So we will perform our novel texture coordinates adjustment scheme to stitch these textures as seamlessly as possible. More specifically, we consider the texture coordinates $t_i \in \mathbb{R}^6$ for each triangle can be moved freely in the input image (texture) space. Starting from the projective texture coordinates, we add additional constraints on these values to remove the seams. The goal of our scheme is to optimize texture coordinates to satisfy all constraints on a per image basis. This can be done in two steps. The first step is the *local phase*, in which the *constrained* texture coordinates are computed for selected triangle. This calculation only depends on the triangle itself and its 1-ring neighbor, therefore they can be done in parallel. The second step is the *global phase*, in which we adjust all the texture coordinates in one image to satisfy, as much as possible, all the constraints as well as the original projective texture coordinates. We will defer the details of this step to Section 5.2.

5.1 Local Optimization

We first explain exactly how the triangles are selected and what kind of constraints are put on their texture coordinates in the local optimization. As we have shown in Fig. 2, there are two possible cases for misalignment. The first is between two different source images. The constraint to remove this inter-image misalignment is called the *inter-image constraint*. The second is within a single image, caused by depth discontinuities, the corresponding constraint to remove this type of misalignment is called the *intra-image constraint*.

5.1.1 Inter-image Constraints

Triangles whose neighboring triangles have different source images are subject to the inter-image constraints. Our inter-image constraint is similar to previous texture optimization techniques (e.g., [7]). It is designed to minimize mismatches between two input images' contents along the shared triangle edges. The matching cost can be defined as:

$$E_{\text{edge}}(F_i, F_j) = \int_{F_i \cap F_j} \mathcal{M}(\phi_{F_i}^{l_i}(x), \phi_{F_j}^{l_j}(x)) dx \quad (2)$$

where $\phi_{F_i}^{l_i}$ is a projection operator of F_i onto image I_{l_i}, and $\mathcal{M}(.,.)$ is a distance metric. Note that $E_{\text{edge}}(F_i, F_j)$ should be zero when F_i and F_j are the projected to the same image, e.g., $l_i = l_j$. We define the distance metric as the sum of color difference and gradient difference, that is

$$\mathcal{M}(x_i, x_j) = \alpha \|I_{l_i}(x_i) - I_{l_j}(x_j)\| + (1-\alpha)\|G_{l_i}(x_i) - G_{l_j}(x_j)\| \quad (3)$$

where x_i and x_j are the 2D texture coordinates in image l_i and l_j of one 3D point on the shared edge of corresponding triangle F_i and F_j, respectively. For the color difference, we have evaluated several different color spaces, such as RGB or LUV, they have similar performance. So we decide to use RGB in all of our experiments. α is a weighting factor, which is set to 0.9 in the implementation empirically.

To get visually matched seams, we search a proper texture coordinates shift $(\Delta x_i, \Delta x_j)$ to minimize $E_{\text{edge}}(F_i, F_j)$. The objective function is in form of

$$E_{\text{opt}} = \kappa \int_{F_i \cap F_j} \mathcal{M}(\phi_{F_i}^{l_i}(x) + \Delta x_i, \phi_{F_j}^{l_j}(x) + \Delta x_j) dx \quad (4)$$

where κ is a texture movement indicator, which is used to penalize unnecessary movement caused color space noise and same translation along the edge. It is defined as

$$\kappa = (1 + \|\Delta x_i\|^2)(1 + \|\Delta x_j\|^2) \quad (5)$$

To optimize Eq. (4), the naive way is to greedily search all the $\|\Delta x\| < \Delta x_{max}$. However, this can be improved by a random line search alternately on Δx_i and Δx_j.

After minimize Eq. (4), we set $\tilde{x}_i = x_i + \Delta x_i$ as an inter-image constraint for the original texture coordinates x_i and set $\tilde{\omega} = \Delta E_{\text{opt}}/E_{\text{opt}}$ as the corresponding constraint weight. Note that this inter-image constraint is solved on a per-triangle basis. All triangles in the current input images can be solved in parallel. We use GPU to accelerate this task.

5.1.2 Intra-image Constraints

The intra-image misalignment is caused by the inconsistency between depth boundaries and color edges. Therefore we first identify potential intra-image misalignment by running edge detection on both the color image and the synthetic depth map that is generated by projecting the 3D model onto the image. For any triangle that (1) has at least one edge on the depth boundaries and (2) is close to a color edge within the threshold $\upsilon = 20$ pixels, we mark it as a candidate to apply the intra-image constraint. Denote the 2D texture coordinate in image space of two endpoints of one triangle edge on the depth boundary as p_0 and p_1. We move both endpoints along the normal direction of $\overrightarrow{p_0 p_1}$ until they intersect a color edge, where we record the new texture coordinates as $\tilde{p}_0$ and $\tilde{p}_1$. Thus, $\tilde{p}_0$ and $\tilde{p}_1$ form an intra-image constraint for p_0 and p_1 respectively. The corresponding constraint weights are set to be inversely proportional to the distance as $\tilde{\omega} = \exp(-(\|\tilde{p}_0 - p_0\|^2 + \|\tilde{p}_1 - p_1\|^2)/2\phi^2)$. ϕ is a weighting factor, which is set to 4.0 empirically. Fig. 4 shows the ability of our method to remove artifacts caused by intra-image misalignment.

Fig. 4. From left to right, before and after intra-image constraint applied. The red color on the wall in (a) is due to misalignment of the images.

5.2 Global Optimization

In global phase, we optimize the texture coordinates of the model in each image to satisfy all proposed constraints. As global optimization is performed in one image at one time, we can now drop l_i and denote all the texture coordinates as $T = \{t_1, t_2, ..., t_n\}$. The cost function for this global phase is defined as

$$E(\tilde{T}) = \frac{\lambda}{2}\|\mathbf{L}\tilde{T} - \rho\|_F^2 + \sum_k C_k(\tilde{T}, T_k) \tag{6}$$

where $\mathbf{L}$ is the Laplacian matrix using cotangent weight, $\rho = \nabla^2(T)$ is the divergence of original projective texture coordinates gradient. The first term basically requires that $\tilde{T}$ should be similar to the original T. λ is a user-defined weighting factor with typical value 0.5.

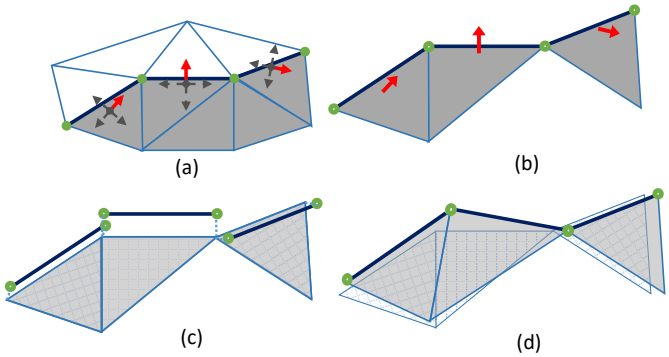


Fig. 5. Local/global texture coordinates adjustment. (a) Each triangle on the share edges yields one constraint, which leads a searching for a proper texture coordinate shifting to remove seam. (b) Triangles need local adjustment under the constraints. (c) The constraints are performed independently in the local phase (the texture coordinates of the share edges are optimized separately). (d) All constraints are satisfied and all rest texture coordinates are adjusted in the global phase.

The second term $C_k(\tilde{T}, T_k)$ is the additional constraint. The subscript k is to differentiate different types of constraints. In general, a constraint has the form of:

$$C_k(\tilde{T}, T_k) = \frac{\omega_k}{2}\|\mathbf{A_k}\mathbf{S_k}\tilde{T} - \mathbf{B_k}T_k\|_F^2 \tag{7}$$

where T_k is an auxiliary variable representing the expected texture coordinates ($\tilde{x}$ in Section 5.1.1 or $\tilde{p}$ in Section 5.1.2) under the current constraint. It should be noted that not all texture coordinates have constraints. For those that are not subject to constraint k, its values in T_k will be zero. $\mathbf{A_k}$ and $\mathbf{B_k}$ are the weighting matrices for T_k, which are updated in every iteration from corresponding weight $\tilde{\omega}$ in Section 5.1.1 and 5.1.2 to indicate the confidence for the current computed T_k. $\mathbf{S_k}$ is the selection matrix. It is a diagonal matrix. Its diagonal elements are one if the corresponding texture coordinates in T_k is not zero (e.g., the constraint is effective), otherwise they will be zero. Finally ω_k is non-negative weight balancing different constraints with typical value 1.0;

We will solve Eq. (6) in an iterative fashion. First we keep $\tilde{T}$ fixed, and solve each constraint for its own set of auxiliary variables T_c. This minimization can be performed independently and in parallel. We will discuss specific constraint types in Section 5.1. Then we will optimized the entire cost function by plugging in the fixed T_j, which leads to a linear system:

$$\lambda\mathbf{L}^\mathbf{T}\mathbf{L} + \sum_j \omega_j \mathbf{S_j^T}\mathbf{A_j^T}\mathbf{A_j}\mathbf{S_j}\tilde{T} = \lambda\mathbf{L}^\mathbf{T}\rho + \sum_j \omega_j \mathbf{S_j^T}\mathbf{A_j^T}\mathbf{A_j}\mathbf{B_j}T_j \tag{8}$$

Eq. (8) is a sparse system, which can be solved using any standard sparse solver. One detail to mention is that the texture coordinates are solved on a per-triangle basis. So a single vertex may have different texture coordinates. To ensure texture continuity within an input image, we add additional equations to the linear system to constrain all the texture coordinates for the same vertex to be equal.

The procedure of our local/global approach is described in Algorithm 1. The initial value of T is computed using projective mapping and image labeling. Note that we only need to calculate $\mathbf{L}$ and ρ once through the iterative process. Usually one a few iterations are needed to solve for one image. After one image is solved, we move on to the next image until all triangles have their

Algorithm 1 Texture Solver

```
loop sloverIteration times
    if sloverIteration == 1 then
        T ← initialValue
    else
        T ← T̃
    end if
    for all constraints j do
        T_j = ConstraintCalculate(C_j, T) (In parallel)
    end for
    T̃ = SolveLinearSystem(T, T_0, T_1, T_2, ...T_j)
end loop
```

optimized texture coordinates. An intuitive explanation of such a procedure is shown in Fig. 5. Fig. 6 visualizes the amount of texture coordinates translation after such a procedure using the example shown in Fig. 1.

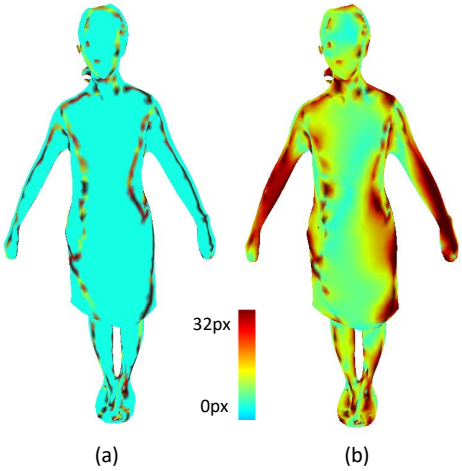


Fig. 6. Texture coordinates shift amount after local phase (a) and global phase (b) in image space optimization, respectively.

5.3 Color Blending

After texture coordinates optimization, misalignment errors are mostly gone. However, color difference between image boundaries can still be visible. The difference is caused by multiple factors, including automatic camera gain, and illumination changes (such as scanning an object on a turn-table). We address these problem using a global color adjustment [8] followed by a local Poisson Blending [14], [26].

6 Experimental Results

We have implemented our method with GPU acceleration and validated it in a series of experiments. All experiments are carried out on a commodity computer (CPU:I7-2600, RAM:16GB, and NVIDIA GeForce GTX 1080).

We first evaluated our method with synthetic data. We generate a cylindrical surface with two input images which are shifted to demonstrate misalignment. Fig. 7 (a) shows result after image labeling using method [8]. Only optimizing the selection of images for texture coherence cannot handle this challenging case since overlap of these two images are small. The results of our method are shown in Fig. 7 (b) and (c), for the local and global

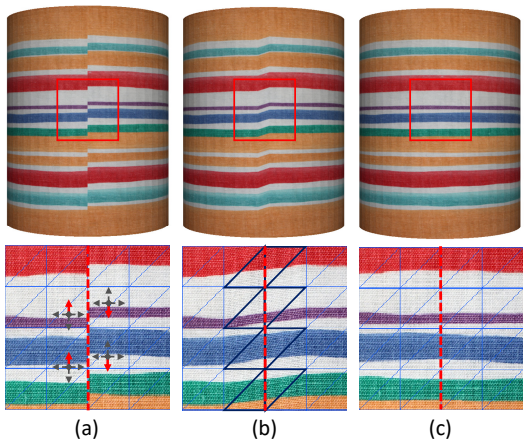


Fig. 7. Synthetic cylinder. (a) Result after image labeling. (b) Result after local optimization (constraints applied) of our method. (c) Final result of our method.

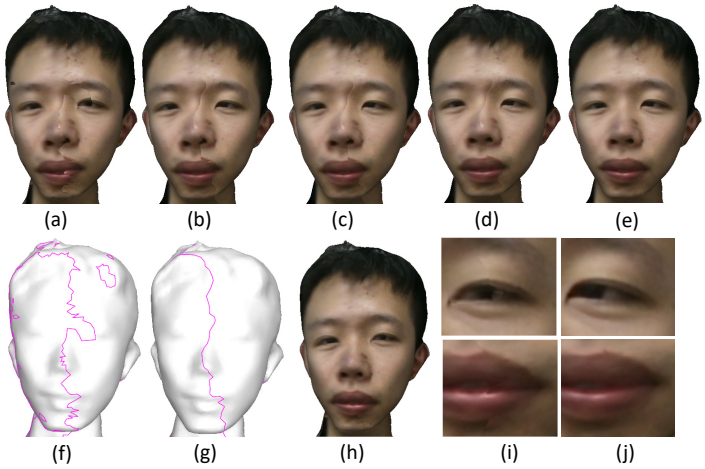


Fig. 8. Texturing a human face. (a) Result using projective mapping (texture seams shown in (f)). (b) Result after image labeling (texture seams shown in (g)). (c)∼(d)Result after first and second iteration of image space optimization in our method. (e) Final result of our method (close views are shown in (j)). (h) Result of Gal et al.'s method (close views are shown in (i)).

optimization phase respectively. In the local phase, sharp seams are stitched but image contents are distorted. After the global phase, seamless and undistorted textures are generated.

We next illustrate the process of our algorithm by texturing a human face. The face model is produced by KinectFuion [1]. We select only two registered images to evaluate our method. The first row of Fig. 8 shows the progressive improvement of each step in our pipeline. We also evaluate Gal et al.'s method [7] with the same setting. Even though some details such as eyes and lips (Fig. 8 (h),(i)) are different with our methods (Fig. 8 (e),(j)), both methods could effectively remove seams and generate plausible results comparing to the naive projective texture mapping (Fig. 8 (a)) and image labeling optimization (Fig. 8 (b)).

Our next data set is acquired with four structured light scanners with high-resolution textures, designed for full-body scanning. While the geometry and calibration is fairly accurate from this set, human subjects are prone to encounter slight movement, in particular the extended arms, during the scanning process, which lasts a few seconds. In order to create a seamless geometric model,

Model	Input Images	Mesh Faces	View Select Time	Local Adjust Faces	Local Adjust (CPU/GPU)	Global Warp Time	Blending Time	Gal et al.'s Time	Total Time (CPU/GPU)
Woman	4@5184×3456	10k	0.02s	1.34k	39.66s / 0.72s	0.14s	4.69s	260s	44.51s / 5.57s
Man	4@5184×3456	10k	0.02s	1.22k	37.18s / 0.68s	0.13s	4.73s	254s	42.06s / 5.56s
Face	2@1920×1080	6k	0.01s	0.48k	16.47s / 0.43s	0.10s	1.27s	132s	17.85s / 1.81s
Fountain	6@1920×1080	25k	0.05s	3.31k	59.19s / 0.93s	0.27s	2.12s	398s	61.63s / 3.37s
Copyroom	11@1920×1080	30k	0.14s	3.64k	65.11s / 1.15s	0.31s	3.13s	495s	68.69s / 4.73s
Apartment	73@640×480	50k	1.26s	6.46k	113.07s / 4.58s	1.42s	4.43s	1248s	120.18s / 11.69s

TABLE 1
Computational time. Note that Gal et al.'s method is evaluated with our implementation using same hardware and same image space search range(± 32 pixels), and all CPU code is evaluated without any optimization, such as multi-threading.

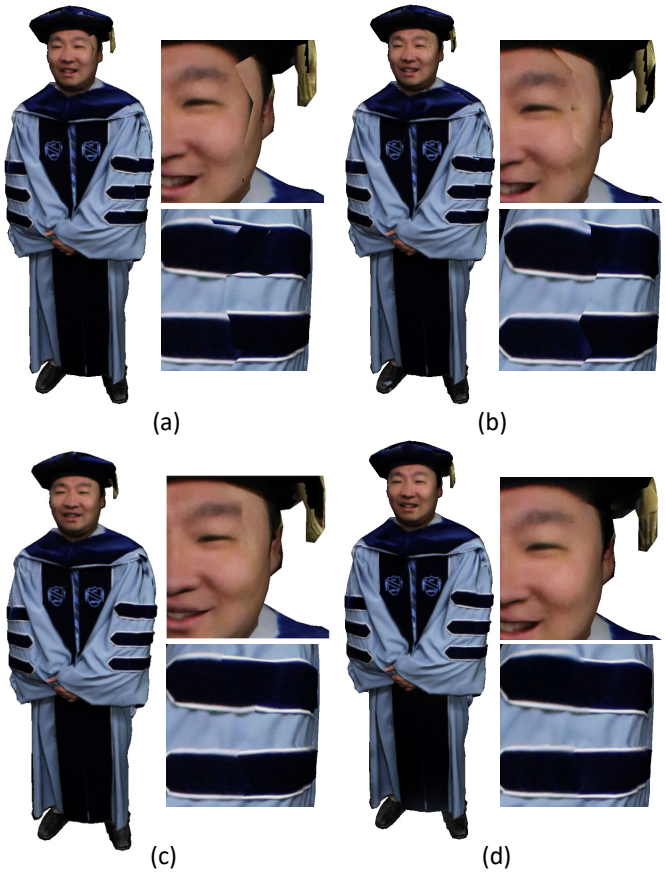


Fig. 9. Textured man models and corresponding close views using projective mapping (a), after image labeling optimization (b), using Gal et al.'s method (c) and using our method (d), respectively.

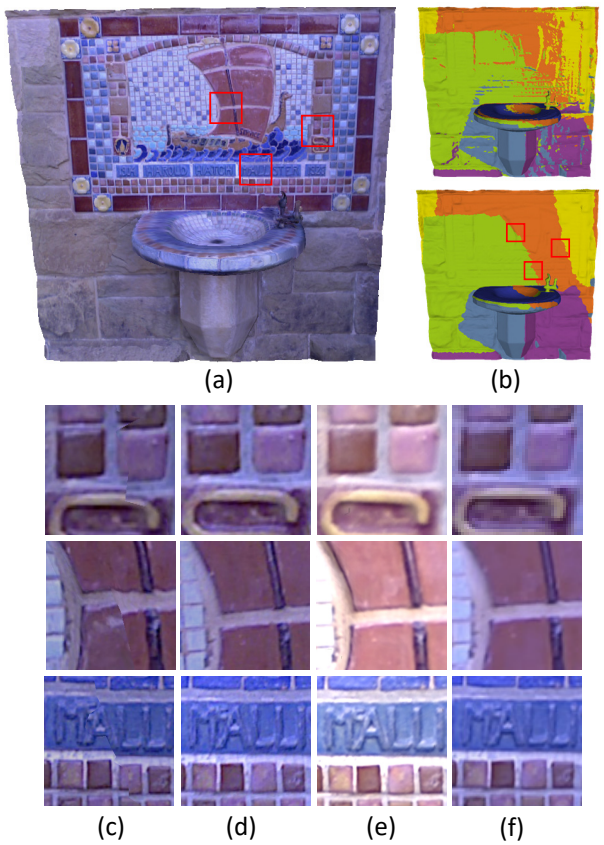


Fig. 10. Fountain dataset from [18]. (a) Textured model using our method. (b) From top to bottom, color-coded image source before and after image labeling optimization. Close views (the red box in (a),(b)) of textured models after image labeling optimization (c), after image space optimization of our method (d), using Zhou et al.'s method [18] (e) and using Bi et al.'s method [19] (f), respectively. Note that the results with different color tones in (e) are directly rendered using Zhou et al.'s published model.

non-rigid alignment is used to create the 3D model, which caused mis-registration with respect to the texture cameras. Fig. 9 shows that our method is able to overcome the mis-registration and create a seamless texture mapping.

Next we apply our texture optimization approach to various datasets acquired with a hand-held Kinect camera. Fig. 11 shows a copy room. The $4m \times 2m \times 2m$ scene is scanned using the voxel hashing approach [3]. There are over five hundred raw images, out of which 11 images are selected. We can see significant improvement of our method compared to voxel coloring and Waechter et al.'s [9] method.

Fig. 10 shows comparison with a voxel-color optimization approach of Zhou et al. [18] and a patch-based texture optimization method of Bi et al. [19] using a fountain data set adopted from [18]. Different with image labeling and texture coordinates optimization related methods [7], [9], those two methods focus on optimizing color or texture patches associated with the model vertices. Thus, a dense model and more images are required by the optimization, which makes them more complex in both computation and storage. It is reported [18] that a mesh with 16 million triangles and 33 images are used to generate the results shown in Fig. 10 (e), with two hundreds of seconds computation time and over hundreds of megabytes to store the model. Bi et al.'s

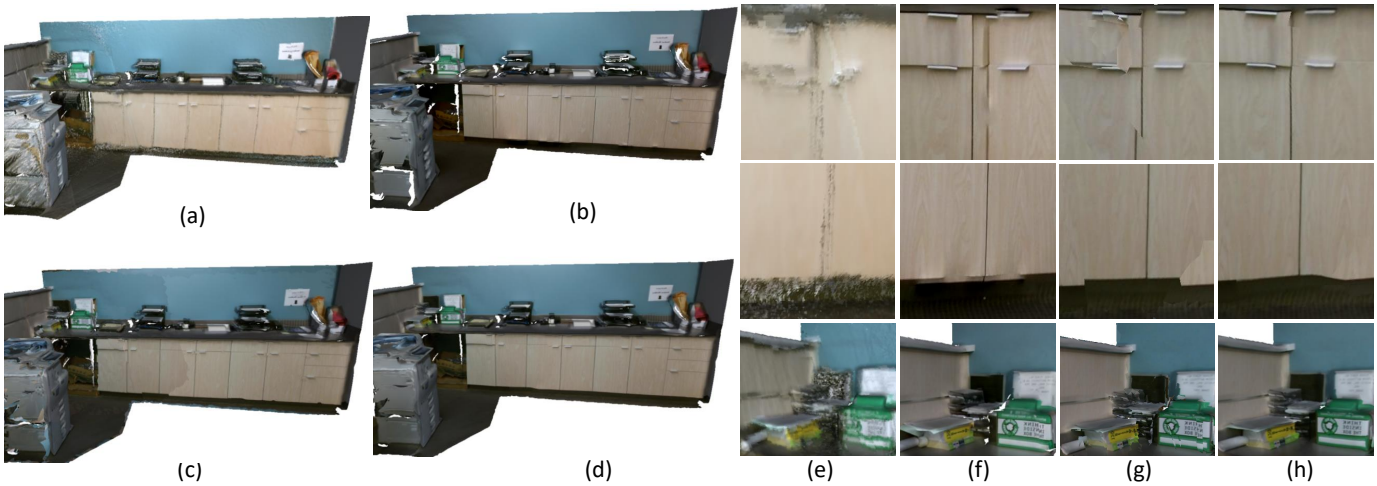


Fig. 11. Texturing a copy room scanned by Kinect. (a) Colorized model using vertex color according to [3] (close views are shown in (e)). (b) Textured model optimized using Waechter et al.'s method [9] (close views are shown in (f)). Textured model after image labeling optimization (c) and image space optimization (d) in our method (close views are shown in (g),(h)), respectively.

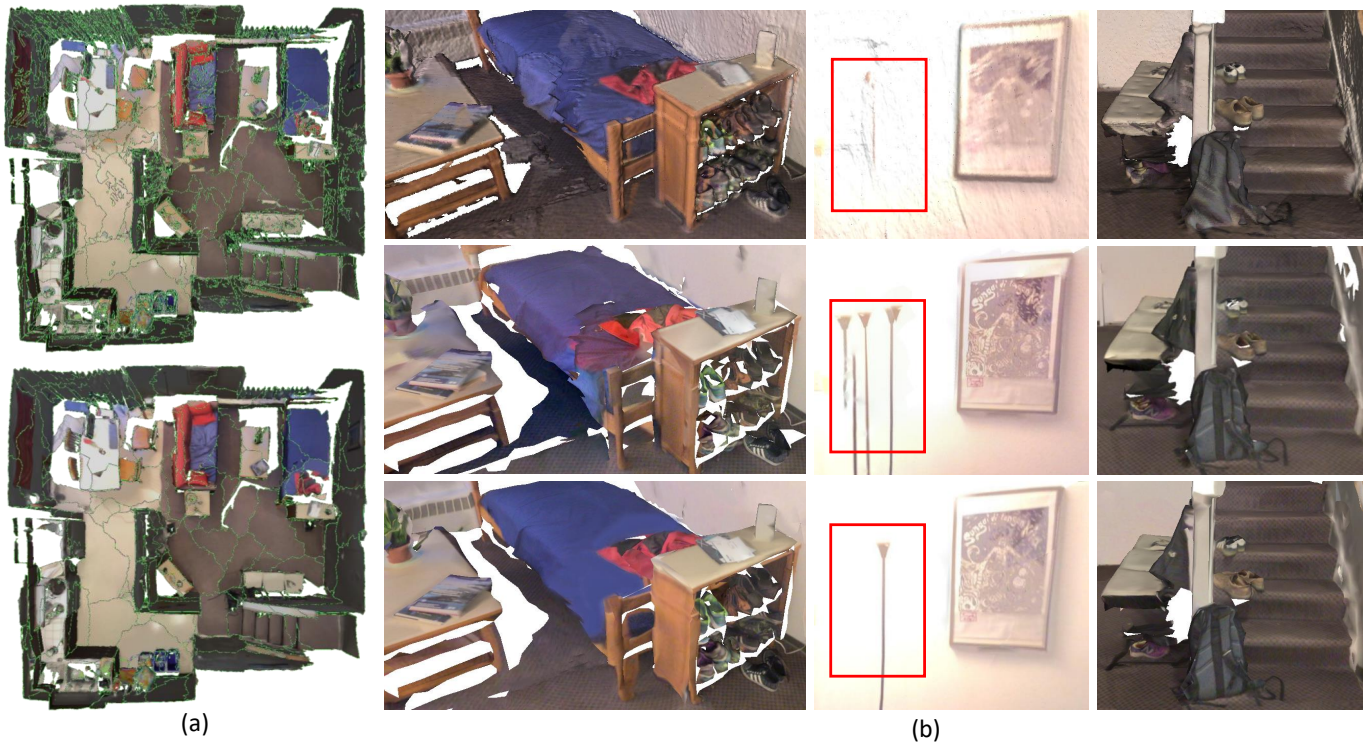


Fig. 12. Texturing a Kinect scanned large scale scene, an apartment, curtesy of [4]. (a) From top to bottom, model with texture seams before and after image labeling. (b) Close-up views, from top to bottom, colorized model by vertex color [4], textured model using Waechter et al.'s method [9] and our method, respectively.

method can also produce a comparable result shown in Fig. 10 (f) using the same setting, however, its computation and storage complexities are also in the same order with [18]. On the contrary, we use only 6 images and a simple mesh with 25k triangles to evaluate our method with result shown in Fig. 10 (a,d). With the novel image space optimization, our method could remove the artifacts along the texture boundaries after image labeling (shown in Fig. 10 (b)). It should be emphasized that our method need only 3.37s to optimize and 5 megabytes to store the result model with textures.

Fig. 12 shows that we apply our method to a much larger apartment scene found in [4]. We selected 73 key images out of the original 8k images sequence. Even the images are at a low resolution of 640×480, our model has much more details.

6.1 Speed Analysis

We summarize the computation time and various statistics in Table 1. We compare it to Gal's method [7], which is the only method that can generate similar results to ours. We have implemented its method in C++, same as our implementation. Note that all Gal's evaluations use five pyramid level (max ± 32 pixels translation) and solve 3 MRF iterations at each level.

As can be seen from Table 1, our method achieves 50∼100 times speed up over Gal's method when using GPU, even the full CPU version is 6∼10 times faster. It should be emphasized that Gal's method is hard to implement on GPU due to its use of graph-cut to solve the entire problem. We could not find a general GPU-based graph-cut implementation. A GPU implementation reported in [27] is valid only for a special graph on a regular grid (e.g., a node has a fixed four or eight neighbors), which cannot be applied in our case in which the mesh connectivity is irregular.

The speed up of our method comes from (1) significantly reducing the label space of graph cut. Graph-cut is only used in the view selection stage, with the number of labels per node in the order of tens, instead of hundreds as in [7], (2) processing selected triangles, typically only 10% of triangles are adjusted in the adjustment stage; and (3) turning the global graph-cut-based optimization into a linear system. These improvements caused the six-time speed-up in CPU. When the second adjustment stage, which is the most time-consuming part, is carried out in GPU, the speed-up is even more dramatic at 50∼100×.

7 Conclusion and Future Work

Texturing plays an important role in a model's appearance. In practice, a perfect texture is always hard to achieve since inaccuracy in raw data, registration, and calibration parameters. These errors are particularly significant in casual 3D scanning with low cost sensors such as the MS Kinect or Intel RealScense. Given a 3D model and calibrated images, we present a novel framework to optimizes the initial projective texture coordinates to generate seamless textured models, in the presence of substantial inaccuracy in the input. Experiments demonstrate the capability of our framework to produce fine texture maps with an average speedup about 50∼100 times compared to previous competing methods. We hope our method can lead to broad dissemination of 3D modeling technology, allowing everyone to create quality textured 3D models with a single low-cost depth sensor.

Looking into the future, we would like to further improve our method for real-time online texture mapping. This kind of capability is particularly useful for many AR/VR applications, such as live 3D broadcasting. Our current formulation requires a triangle mesh. Since current the state-of-the-art live 3D reconstruction methods are all based on a volumetric presentation, we would also like to extend our method to work with a voxel grid directly.

References

[1] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136.

[2] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison *et al.*, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 559–568.

[3] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 169, 2013.

[4] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration," *arXiv preprint arXiv:1604.01093*, 2016.

[5] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 343–352.

[6] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger, "Volumedeform: Real-time volumetric non-rigid reconstruction," *arXiv preprint arXiv:1603.08161*, 2016.

[7] R. Gal, Y. Wexler, E. Ofek, H. Hoppe, and D. Cohen-Or, "Seamless montage for texturing models," in *Computer Graphics Forum*, vol. 29, no. 2. Wiley Online Library, 2010, pp. 479–486.

[8] V. Lempitsky and D. Ivanov, "Seamless mosaicing of image-based texture maps," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2007, pp. 1–6.

[9] M. Waechter, N. Moehrle, and M. Goesele, "Let there be color! large-scale texturing of 3d reconstructions," in *European Conference on Computer Vision*. Springer, 2014, pp. 836–850.

[10] L. Wang, S. B. Kang, R. Szeliski, and H.-Y. Shum, "Optimal texture map reconstruction from multiple views," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–347.

[11] A. Baumberg, "Blending images for texturing 3d models." in *BMVC*, vol. 3. Citeseer, 2002, p. 5.

[12] J. Totz, A. J. Chung, and G.-Z. Yang, "Patient-specific texture blending on surfaces of arbitrary topology," in *Workshop on Augmented environments for Medical Imaging and Computeraided Surgery (AMI-ARCS), London, UK*. Citeseer, 2009, pp. 78–85.

[13] C. Rocchini, P. Cignoni, C. Montani, and R. Scopigno, "Multiple textures stitching and blending on 3d objects," in *Rendering Techniques 99*. Springer, 1999, pp. 119–130.

[14] A. Dessein, W. A. Smith, R. C. Wilson, and E. R. Hancock, "Seamless texture stitching on a 3d mesh by poisson blending in patches," in *2014 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 2031–2035.

[15] C. Soler, M.-P. Cani, and A. Angelidis, "Hierarchical pattern mapping," *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3, pp. 673–680, 2002.

[16] M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. De Aguiar, N. Ahmed, C. Theobalt, and A. Sellent, "Floating textures," in *Computer Graphics Forum*, vol. 27, no. 2. Wiley Online Library, 2008, pp. 409–418.

[17] E. Aganj, P. Monasse, and R. Keriven, "Multi-view texturing of imprecise mesh," in *Asian Conference on Computer Vision*. Springer, 2009, pp. 468–476.

[18] Q.-Y. Zhou and V. Koltun, "Color map optimization for 3d reconstruction with consumer depth cameras," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 155, 2014.

[19] S. Bi, N. K. Kalantari, and R. Ramamoorthi, "Patch-based optimization for image-based texture mapping," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, vol. 36, no. 4, 2017.

[20] K. Zhou, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum, "Texturemontage," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 1148–1155, 2005.

[21] R. Schmidt, C. Grimm, and B. Wyvill, "Interactive decal compositing with discrete exponential maps," *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3, pp. 605–613, 2006.

[22] Y. Tzur and A. Tal, "Flexistickers: photogrammetric texture mapping using casual images," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3. ACM, 2009, p. 45.

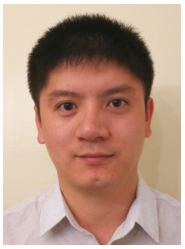
[23] V. Kraevoy, A. Sheffer, and C. Gotsman, *Matchmaker: constructing constrained texture maps*. ACM, 2003, vol. 22, no. 3.

[24] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe, "Texture mapping progressive meshes," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 409–416.

[25] A. Delong, A. Osokin, H. N. Isack, and Y. Boykov, "Fast approximate energy minimization with label costs," *International journal of computer vision*, vol. 96, no. 1, pp. 1–27, 2012.

[26] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," in *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3. ACM, 2003, pp. 313–318.

[27] V. Vineet and P. Narayanan, "Cuda cuts: Fast graph cuts on the gpu," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*. IEEE, 2008, pp. 1–8.



Wei Li is pursuing his PhD degree in Nanjing University of Aeronautics and Astronautics. During 09/2015 to 09/2017, he was a visiting Ph.D. student at the University of Kentucky. Now, he is also an intern of the Baidu Research. His recent research interests focus on computer vision and graphics. He is particularly interested in physically based modeling/simulation, 3D modeling and rendering, 3D display.

Huajun Gong is currently a professor in College of Automation Engineering, Nanjing University of Aeronautics and Astronautics. He received his PhD degree from Nanjing University of Aeronautics and Astronautics in 1999. His main research interests include control and navigation of aerocraft, machine vision and 3D display.

Ruigang Yang is the Chief Scientist of 3D vision at Baidu Research and a full professor of computer science at the University of Kentucky (on leave). He obtained his PhD degree from University of North Carolina at Chapel Hill and his MS degree from Columbia University. His research interests span over computer graphics and computer vision, in particular in 3D reconstruction and 3D data analysis. He has published over 100 papers, which, according to Google Scholar, has received close to 10000 citations with an h-index of 47 (as of 2017). He has received a number of awards, including US NSF Career award in 2004 and the Deans Research Award at the University of Kentucky in 2013.