Initialization and Alignment for Adversarial Texture Optimization

Xiaoming Zhao(✉), Zhizhen Zhao, and Alexander G. Schwing

University of Illinois, Urbana-Champaign, USA
{xz23,zhizhenz,aschwing}@illinois.edu
https://xiaoming-zhao.github.io/projects/advtex_init_align

Abstract. While recovery of geometry from image and video data has received a lot of attention in computer vision, methods to capture the texture for a given geometry are less mature. Specifically, classical methods for texture generation often assume clean geometry and reasonably well-aligned image data. While very recent methods, *e.g.*, adversarial texture optimization, better handle lower-quality data obtained from hand-held devices, we find them to still struggle frequently. To improve robustness, particularly of recent adversarial texture optimization, we develop an explicit initialization and an alignment procedure. It deals with complex geometry due to a robust mapping of the geometry to the texture map and a hard-assignment-based initialization. It deals with misalignment of geometry and images by integrating fast image-alignment into the texture refinement optimization. We demonstrate efficacy of our texture generation on a dataset of 11 scenes with a total of 2807 frames, observing 7.8% and 11.1% relative improvements regarding perceptual and sharpness measurements.

Keywords: Scene analysis · Texture reconstruction

1 Introduction

Accurate scene reconstruction is one of the major goals in computer vision. Decades of research have been devoted to developing robust methods like 'Structure from Motion,' 'Bundle Adjustment,' and more recently also single view reconstruction techniques. While reconstruction of geometry from image and video data has become increasingly popular and accurate in recent years, recovered 3D models remain often pale because textures aren't considered.

Given a reconstructed 3D model of a scene consisting of triangular faces, and given a sequence of images depicting the scene, texture mapping aims to find for each triangle a suitable texture. The problem of automatic texture mapping has been studied in different areas since late 1990 and early 2000. For instance, in the graphics community [12,29,30], in computer vision [27,43,45], architecture [19]

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-25066-8_34.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
L. Karlinsky et al. (Eds.): ECCV 2022 Workshops, LNCS 13803, pp. 587–604, 2023.
https://doi.org/10.1007/978-3-031-25066-8_34

and photogrammetry [14]. Many of the proposed algorithms work very well in a controlled lab-setting where geometry is known perfectly, or in a setting where accurate 3D models are available from a 3D laser scanner.

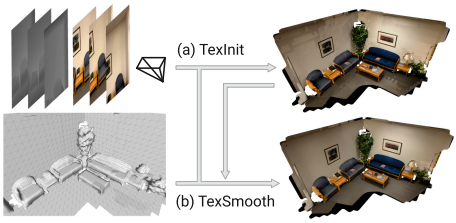


Fig. 1. We study texture generation given RGBD images with associated camera parameters as well as a reconstructed mesh. **(a) `TexInit`**: we initialize the texture using an assignment-based texture generation framework. **(b) `TexSmooth`**: a data-driven adversarial loss is utilized to optimize out artifacts incurred in the assignment step.

However, applying texture mapping techniques to noisy mesh geometry obtained *on the fly* from a recent LiDAR equipped iPad reveals missing robustness because of multiple reasons: 1) images and 3D models are often not perfectly aligned; 2) 3D models are not accurate and the obtained meshes aren't necessarily manifold-connected. Even recent techniques for mesh flattening [41] and texturing [16,24,54] result in surprising artifacts due to streamed geometry and pose inaccuracies as shown later.

To address this robustness issue we find equipping of the recently-proposed adversarial texture optimization technique [24] with classical initialization and alignment techniques to be remarkably effective. Without the added initialization and alignment, we find current methods don't produce high-quality textures. Concretely and as illustrated in Fig. 1, we aim for texture generation which operates on a sequence of images and corresponding depth maps as well as their camera parameters. Moreover, we assume the 3D model to be given and fixed. Importantly, we consider a streaming setup, with all data obtained *on the fly*, and not further processed, *e.g.*, via batch structure-from-motion. The setup is ubiquitous and the form of data can be acquired easily from consumer-grade devices these days, *e.g.*, from a recent iPad or iPhone [1]. We aim to translate this data into texture maps. For this, we first flatten the triangle mesh using recent advances [41]. We then use a Markov Random Field (MRF) to resolve overlaps in flattened meshes for non-manifold-connected data. In a next step we determine the image frame from which to extract the texture of each mesh triangle using a simple optimization. We refer to this as `TexInit`, which permits to obtain a high-quality initialization for subsequent refinement. Next we address inaccuracies in camera poses and in geometry by automatically shifting images using the result of a fast Fourier transformation (FFT) [6]. The final optimized texture is obtained by integrating this FFT-alignment component into adversarial optimization [24]. We dub this stage `TexSmooth`. The obtained texture can be used in any 3D engine for downstream applications.

To study efficacy of the proposed framework we acquire 11 complex scenes using a recent iPad. We establish accuracy of the proposed technique to generate and use the texture by showing that the quality of rendered views is superior to prior approaches on these scenes. Quantitatively, our framework improves prior work by 7.8% and 11.1% relatively with respect to perceptual (LPIPS [53]) and sharpness (S_3 [47]) measurements respectively. Besides, our framework improves over baselines on ScanNet [11], demonstrating the ability to generalize.

2 Related Work

We aim for accurate recovery of texture for a reconstructed 3D scene from a sequence of RGBD images. For this, a variety of techniques have been proposed, which can be roughly categorized into four groups: 1) averaging-based; 2) warping-based; 3) learning-based; and 4) assignment-based. Averaging-based methods find all views within which a point is visible and combine the color observations. Warping-based approaches either distort or synthesize source images to handle mesh misalignment or camera drift. Learning-based ones learn the texture representation. Assignment-based methods attempt to find the best view and 'copy' the observation into a texture. We review these groups next:

Averaging-Based: Very early work by Marshner [29] estimates the parameters of a bidirectional reflectance distribution function (BRDF) for every point on the texture map. To compute this estimation, all observations from the recorded images where the point is visible are used. Similar techniques have been investigated in subsequent work [9].

Similarly, to compute a texture map, [30] and [12] perform a weighted blending of all recorded images. The weights take visibility and other factors into account. The developed approaches are semi-automatic as they require an initial estimate of the camera orientations which is obtained from interactively selected point correspondences or marked lines. Multi-resolution textures [33], face textures [36] and blending [8,31] have also been studied.

Warping-Based: Aganj *et al.* [4] morph each source image to align to the mesh. Furthermore, [23,54] propose to optimize camera poses and image warping parameters jointly. However, this line of vertex-based optimization has stringent requirements on the mesh density and cannot be applied to a sparse mesh. More recently, Bi *et al.* [10] follow patch-synthesis [7,50] to re-synthesize aligned target images for each source view. However, such methods require costly multiscale optimization to avoid a large number of local optima. In contrast, the proposed approach does not require those techniques.

Learning-Based: Recently, learning-based methods have been introduced for texture optimization. Some works focus on specific object and scene categories [18,40] while we do not make such assumptions. Moreover, learned rep-

resentations, *e.g.*, neural textures, have also been developed [5,44,46]. Meanwhile, generative models are developed to synthesize a holistic texture from a single image or pattern [21,32] while we focus on texture reconstruction. AtlasNet [20] and NeuTex [52] focus on learning a 3D-to-2D mapping, which can be utilized in texture editing, while we focus on reconstructing realistic textures from source images. The recently-proposed adversarial texture optimization [24] utilizes adversarial training to reconstruct the texture. However, despite advances, adversarial optimization still struggles with misalignments. We improve this shortcoming via an explicit high-quality initialization and an efficient alignment module.

Assignment-Based: Classical assignment-based methods operated within controlled environments [14,15,38,39] or utilized special camera rigs [15,19]. These works suggest computing for each vertex a set of 'valid' images, which are subsequently refined by iterating over each vertex and adjusting the assignment to obtain more consistency. Finally, texture data is 'copied' from the images. In contrast, we aim to create a texture in an uncontrolled setting. Consequently, 3D geometry is not accurate and very noisy. Other early work [13,22,28,39,51] focuses on closed surfaces and small-scale meshes, making them not applicable to our setting. More recently, upon finding the best texture independently for each face using cues like visibility, orientation, resolution, and distortion, refinement techniques like texture coordinate optimization, color adjustments, or scores-based optimization have been discussed [3,34,48].

Related to our approach are methods that formulate texture selection using a Markov Random Field (MRF) [16,27,42]. Shu *et al.* [42] suggest visibility as the data term and employ texture smoothness to reduce transitions. Lempitsky *et al.* [27] study color-continuity which integrates over face seams. Fu *et al.* [16] additionally use the projected 2D face area to select a texture assignment for each face. However, noisy geometry like the one shown in Fig. 2, makes it difficult for assignment-based methods to yield high quality results, which we will show later. Therefore, different from these methods, we address texture drift in a data-driven refinement procedure rather than in an assignment stage.

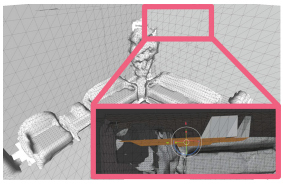


Fig. 2. Noisy geometry. The wall has two layers.

3 Approach

We want to automatically create the texture $\mathcal{T}$ from a set of RGBD images $I = \{I_1, \ldots, I_T\}$, for each of which we also know camera parameters $\{p_t\}_{t=1}^{T}$, *i.e.*, extrinsics and intrinsics. We are also given a triangular scene mesh $M = \{\texttt{Tri}_i\}_{i=1}^{|M|}$, where $\texttt{Tri}_i$ denotes the i-th triangle. This form of data is easily accessible from commercially available consumer devices, *e.g.*, a recent iPhone or iPad.

We construct the texture $\mathcal{T}$ in two steps that combine advantages of assignment-based and learning-based techniques: 1) `TexInit`: we generate a texture initialization $\mathcal{T}_{\text{init}} \in \mathbb{R}^{H \times W \times 3}$ of height H, width W and 3 color channels in an assignment-based manner (Sect. 3.1); 2) `TexSmooth`: we then refine $\mathcal{T}_{\text{init}}$ with an improved data-driven adversarial optimization that integrates an efficient alignment procedure (Sect. 3.2). Formally, the final texture $\mathcal{T}$ is computed via

$$\begin{aligned} \mathcal{T} &= \texttt{TexSmooth}\left(\mathcal{T}_{\text{init}}, \{I_t\}_{t=1}^{T}, \{p_t\}_{t=1}^{T}, M\right), \\ \text{where } \mathcal{T}_{\text{init}} &= \texttt{TexInit}\left(\{I_t\}_{t=1}^{T}, \{p_t\}_{t=1}^{T}, M\right). \end{aligned} \tag{1}$$

We detail each component next.

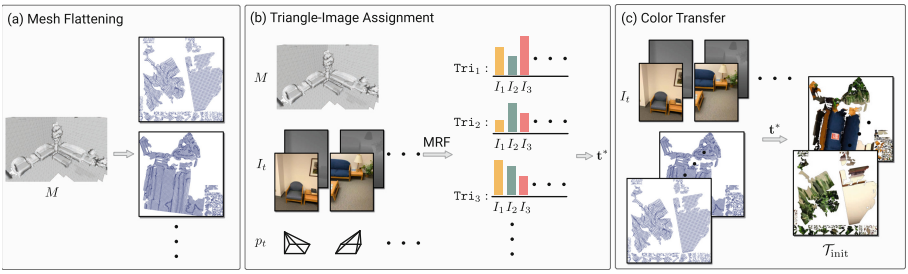


Fig. 3. Texture initialization `TexInit` (Sect. 3.1). **(a)** Mesh flattening: we flatten a 3D mesh into the 2D plane using overlap detection. **(b)** Triangle-image assignment: we develop a simple formulation to compute the triangle-image assignment $\mathbf{t}^*$ from mesh M, frames I_t and camera parameters p_t. We assign frames to each triangle $\texttt{Tri}_i$ based on $\mathbf{t}^*$. **(c)** Color transfer: based on the flattened mesh in **(a)** and the best assignment $\mathbf{t}^*$ from **(b)**, we generate the texture $\mathcal{T}_{\text{init}}$.

3.1 Texture Initialization (`TexInit`)

The proposed approach to obtain the texture initialization $\mathcal{T}_{\text{init}}$ is outlined in Fig. 3 and consists of following three steps: 1) We flatten the provided mesh M. For this we detect overlaps within the flattened mesh, which may happen due to the fact that we operate with general meshes that are not guaranteed to have a manifold connectivity. Overlap detection ensures that every triangle is assigned a unique position in the texture. 2) We identify for each triangle the 'best' texture index $\mathbf{t}^*$. Hereby, 'best' is defined using cues like visibility and color consistency. 3) After identifying the index $\mathbf{t}^* = (t_1^*, \ldots, t_{|M|}^*)$ for each triangle, we create the texture $\mathcal{T}_{\text{init}}$ by transferring for all $(u, v) \in [1, \ldots, W] \times [1, \ldots, H]$ locations in the texture, the RGB data from the corresponding location (a, b) in image I_t.

1) Mesh Flattening: In a first step, as illustrated in Fig. 3 (left), we flatten the given mesh M. For this we use the recently proposed boundary first flattening (BFF) technique [41]. The flattening is fully automatic, with distortion mathematically guaranteed to be as low or lower than any other conformal mapping.

However, despite those guarantees, BFF still requires meshes to have a manifold connectivity. While we augment work by [41] using vertex duplication to circumvent this restriction, flattening may still result in overlapping regions as illustrated in Fig. 4. To fix this and uniquely assign a triangle to a position in the texture, we perform overlap detection as discussed next.

Overlap Detection: Overlap detection operates on flattened and possibly overlapping triangle meshes like the one illustrated in Fig. 4a. Our goal is to assign triangles to different planes. Upon re-packing the triangles assigned to different planes, we obtain the non-overlapping triangle mesh illustrated in Fig. 4b.

In order to not break the triangle mesh at a random position and end up with many individual triangles, *i.e.*, in order to maintain large triangle connectivity, we formulate this problem using a Markov Random Field (MRF). Formally, let the discrete variable $y_i \in \mathcal{Y} = \{1, \ldots, |\mathcal{Y}|\}$ denote the discrete plane index that the i-th triangle $\texttt{Tri}_i$ is assigned to. Hereby, $|\mathcal{Y}|$ denotes the maximum number of planes which is identical to the maximum number of triangles that overlap initially at any one location. We obtain the triangle-plane assignment $y^* = (y_1^*, \ldots, y_{|M|}^*)$ for all $|M|$ triangles by addressing

$$y^* = \arg\max_y \sum_{i=1}^{|M|} \phi_i(y_i) + \sum_{(i,j)\in\mathcal{A}\cup\mathcal{O}} \phi_{i,j}(y_i, y_j), \tag{2}$$

where $\mathcal{A}$ and $\mathcal{O}$ are sets of triangle index pairs which are adjacent and overlapping respectively. Here, $\phi_i(\cdot)$ denotes triangle $\texttt{Tri}_i$'s priority over $\mathcal{Y}$ when considering only its *local* information, while $\phi_{i,j}(\cdot)$ refers to $\texttt{Tri}_i$ and $\texttt{Tri}_j$'s joint preference on their assignments. Equation (2) is solved with belief propagation [17].

Intuitively, by addressing the program given in Eq. (2) we want a different plane index for overlapping triangles, while encouraging mesh M's adjacent triangles to be placed on the same plane. To achieve this we use

$$\phi_i(y_i) = \begin{cases} 1.0, & \text{if } y_i = \min \mathcal{Y}_{i,\text{non-overlap}} \\ 0.0, & \text{otherwise} \end{cases}, \text{ and} \tag{3}$$

$$\phi_{i,j}(y_i, y_j) = \begin{cases} \mathbb{1}\{y_i = y_j\}, & \text{if } (i,j) \in \mathcal{A} \\ \mathbb{1}\{y_i \neq y_j\}, & \text{if } (i,j) \in \mathcal{O} \end{cases}. \tag{4}$$

Here, $\mathbb{1}\{\cdot\}$ denotes the indicator function and $\mathcal{Y}_{i,\text{non-overlap}}$ contains all plane indices where $\texttt{Tri}_i$ has no overlap with others. Intuitively, Eq. (3) encourages to assign the minimum of such indices to $\texttt{Tri}_i$.

As fast MRF optimizers remove most overlaps but don't provide guarantees, we add a light post-processing to manually assign the remaining few overlapping triangles to separate planes. This guarantees overlap-free results. As mentioned before, after having identified the plane assignment y^* for each triangle we use a bin packing to uniquely assign each triangle to a position in the texture. Conversely, for every texture coordinate u, v we obtain a unique triangle index

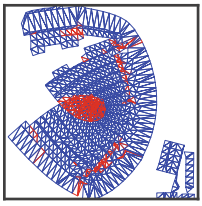
$$i = G(u, v). \tag{5}$$

A qualitative result is illustrated in Fig. 4b. Next, we identify the image which should be used to texture each triangle.

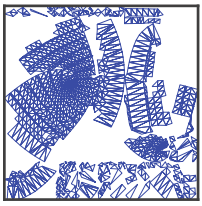
2) Textures from Triangle-Image Assignments: Our goal is to identify a suitable frame I_{t_i}, $t_i \in \{1, \ldots, T+1\}$, for each triangle $\texttt{Tri}_i$, $i \in \{1, \ldots, |M|\}$. Note that the $(T+1)$-th option I_{T+1} refers to an empty texture. We compute the texture assignments $\mathbf{t}^* = (t_1^*, \ldots, t_{|M|}^*)$ using a purely local optimization:

$$\mathbf{t}^* = \underset{\mathbf{t}}{\operatorname{argmax}} \sum_{i=1}^{|M|} \psi_i(t_i). \tag{6}$$

Here ψ_i captures *unary* cues. Note, we also studied *pairwise* cues but did not observe significant improvements. Please see the Appendix for more details. Due to better efficiency, we therefore only consider unary cues. Intuitively, we want



(a) Flattened mesh overlaps.



(b) Overlap-free.

Fig. 4. Flattening. (a) Red triangles indicate where overlap happens. **(b)** The proposed method (Sect. 3.1) resolves this issue while keeping connectivity of areas. (Color figure online)

(a) L2Avg. (b) $\mathcal{T}_{\text{init}}$.

Fig. 5. Initialization comparison. (a) We use PyTorch3D's rendering pipeline [37] to project each pixel of every RGB image back to the texture. The color of each pixel in the texture is the average of all colors that project to it. This texture minimizes the $\mathcal{L}_2$ loss of the difference between the rendered and the ground truth images. We dub it L2Avg. **(b)** $\mathcal{T}_{\text{init}}$ from Sect. 3.1 permits to maintain details. The seam artifacts will be optimized out using `TexSmooth` (Sect. 3.2). Besides over-smoothness, without taking into account misalignments of geometry and camera poses, L2Avg produces texture that overfits to available views, *e.g.*, the sofa's blue colors are painted onto the wall. (Color figure online)

the program given in Eq. (6) to encourage triangle-image assignment to be 'best' for each triangle $\texttt{Tri}_i$. We describe the unary cues to do so next.

Unary Potentials $\psi_i(t_i)$ for each pair of triangle $\texttt{Tri}_i$ and frame I_{t_i} are

$$\psi_i(t_i) = \begin{cases} \psi_i^{\texttt{C}}(t_i), & \text{if } \psi_i^{\texttt{V}}(t_i) = 1 \\ -\infty, & \text{otherwise} \end{cases}, \tag{7}$$

where $\psi_i^{\texttt{V}}(t_i)$ and $\psi_i^{\texttt{C}}(t_i)$ represent validity check and potentials from cues respectively. Concretely, we use

$$\psi_i^{\texttt{V}}(t_i) = \mathbb{1}\{I_{t_i} \in \mathcal{S}_i^{\texttt{V}}\}, \tag{8}$$

$$\psi_i^{\texttt{C}}(t_i) = \omega_1 \cdot \psi_i^{\texttt{C}_1}(t_i) + \omega_2 \cdot \psi_i^{\texttt{C}_2}(t_i) + \omega_3 \cdot \psi_i^{\texttt{C}_3}(t_i), \tag{9}$$

where $\mathcal{S}_i^{\texttt{V}}$ denotes the set of valid frames for $\texttt{Tri}_i$ and $\omega_1, \omega_2, \omega_3$ represent weights for potentials $\psi_i^{\texttt{C}_1}, \psi_i^{\texttt{C}_2}, \psi_i^{\texttt{C}_3}$. We discuss each one next:

- **Validity** ($\texttt{V}$). To assess whether frame I_{t_i} is valid for $\texttt{Tri}_i$, we check the visibility of $\texttt{Tri}_i$ in I_{t_i}. We approximate this by checking visibility of $\texttt{Tri}_i$'s three vertices as well as its centroid. Concretely, we transform the vertices and centroid from world coordinates to the normalized device coordinates of the t_i-th camera. If all vertices and centroid are visible, *i.e.*, their coordinates are in the interval $[-1, 1]$, we add frame I_{t_i} to the set $\mathcal{S}_i^{\texttt{V}}$ of valid frames for triangle $\texttt{Tri}_i$.
- **Triangle area** ($\texttt{C}_1$). Based on a camera's pose p_{t_i}, a triangle's area changes. The larger the area, the more detailed is the information for $\texttt{Tri}_i$ in frame I_{t_i}. We encourage to assign $\texttt{Tri}_i$ to frames I_{t_i} with large area by defining $\psi_i^{\texttt{C}_1}(t_i) = \texttt{Area}_{t_i}(\texttt{Tri}_i)$ and set $\omega_1 > 0$.

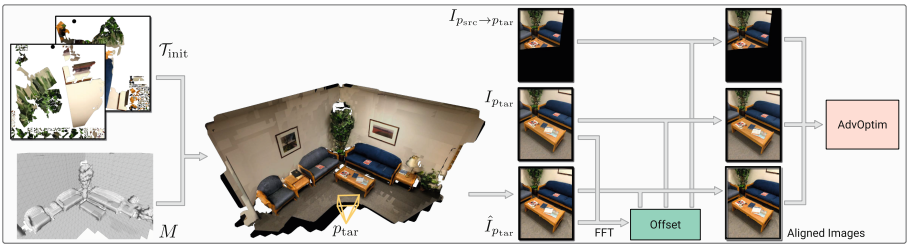


Fig. 6. Texture smoothing $\texttt{TexSmooth}$ (Sect. 3.2). We utilize adversarial optimization ($\texttt{AdvOptim}$) [24] to refine the texture $\mathcal{T}_{\text{init}}$ from Sect. 3.1. Differently: 1) We initialize with $\mathcal{T}_{\text{init}}$. 2) To resolve the issue of misalignment between rendering and ground truth (GT), we integrate an alignment module based on the fast Fourier transform (FFT).

- **Discrepancy between z-buffer and actual depth** ($\mathtt{C}_2$). For a valid frame $I_{t_i} \in \mathcal{S}_i^{\mathsf{v}}$, a triangle's vertices and its centroid project to valid image coordinates. We compute the discrepancy between: 1) the depth from frame I_{t_i} at the image coordinates of the vertices and centroid; 2) the depth of vertices and centroid in the camera's coordinate system. We set $\psi_i^{\mathtt{C}_2}(t_i)$ to be the sum of absolute value differences between both depth estimates while using $\omega_2 < 0$.
- **Perceptual consistency** ($\mathtt{C}_3$). Due to diverse illumination, triangle $\mathtt{Tri}_i$'s appearance changes across frames. Intuitively, we don't want to assign a texture to $\mathtt{Tri}_i$ using a frame that contains colors that deviate drastically from other frames. Concretely, we first average all triangle's three vertices color values across all valid frames, *i.e.*, across all $I_{t_i} \in \mathcal{S}_i^{\mathsf{v}}$. We then compare this global average to the local average obtained independently for the three vertices of every valid frame $I_{t_i} \in \mathcal{S}_i^{\mathsf{v}}$ using an absolute value difference. We require $\omega_3 < 0$.

3) Color Transfer: Given the inferred triangle-frame assignments $\mathbf{t}^*$ we complete the texture $\mathcal{T}_{\text{init}}$ by transferring RGB data from image $I_{t_i^*}$ for $\mathtt{Tri}_i, i \in \{1, \ldots, |M|\}$. For this we leverage the camera pose $p_{t_i^*}$ which permits to transform the texture coordinates (u, v) of locations within $\mathtt{Tri}_i$ to corresponding image coordinates (a, b) in texture $I_{t_i^*}$ via the mapping $F : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, *i.e.*,

$$(a, b) = F(u, v, t_i^*, p_{t_i^*}). \tag{10}$$

Intuitively, given the (u, v) coordinates on the texture in a coordinate system which is local to the triangle $\mathtt{Tri}_i$, and given the camera pose $p_{t_i^*}$ used to record image $I_{t_i^*}$, the mapping F retrieves the image coordinates (a, b) corresponding to texture coordinate (u, v). Using this mapping, we obtain the texture $\mathcal{T}_{\text{init}}$ at location (u, v), *i.e.*, $\mathcal{T}_{\text{init}}(u, v)$, from the image data $I_{t_i^*}(a, b) \in \mathbb{R}^3$ via

$$\mathcal{T}_{\text{init}}(u, v) = I_{t_i^*}(F(u, v, t_i^*, p_{t_i^*})). \tag{11}$$

Note, because of the overlap detection, we obtain a unique triangle index $i = G(u, v)$ for every (u, v) coordinate from Eq. (5). Having transferred RGB data for all coordinates within all triangles results in the texture $\mathcal{T}_{\text{init}} \in \mathbb{R}^{H \times W \times 3}$, which we compare to standard L2 averaging initialization in Fig. 5. We next refine this texture via adversarial optimization. We observe that this initialization $\mathcal{T}_{\text{init}}$ is crucial to obtain high-quality textures, which we will show in Sect. 4.

3.2 Texture Smoothing (`TexSmooth`)

As can be seen in Fig. 5b, the texture $\mathcal{T}_{\text{init}}$ contains seams that affect visual quality. To reconstruct a seamless texture $\mathcal{T}$, we extend recent adversarial

optimization (AdvOptim). Different from prior work [24] which initializes with blank (paper) or averaged (code release[1]) textures, we initialize with $\mathcal{T}_{\text{init}}$. Also, we find AdvOptim doesn't handle common camera pose and geometry misalignment well. To resolve this, we develop an efficient alignment module. This is depicted in Fig. 6 and will be detailed next.

Smoothing with Adversarial Optimization: To optimize the texture, AdvOptim iterates over camera poses. When optimizing for a specific target camera pose p_{tar}, AdvOptim uses three images: 1) the ground truth image $I_{p_{\text{tar}}}$ of the target camera pose p_{tar}; 2) a rendering $\hat{I}_{p_{\text{tar}}}$ for the target camera pose p_{tar} from the texture map $\mathcal{T}$; and 3) a re-projection from another camera pose p_{src}'s ground truth image, which we refer to as $I_{p_{\text{src}} \to p_{\text{tar}}}$. It then optimizes by minimizing an $\mathcal{L}_1$ plus a conditional adversarial loss. However, we find AdvOptim to struggle with alignment errors due to inaccurate geometry. Therefore, we integrate an efficient alignment operation into AdvOptim. Instead of directly using the input images, we first compute a 2D offset $(\Delta h_{p_{\text{tar}}}, \Delta w_{p_{\text{tar}}})$ between $I_{p_{\text{tar}}}$ and $\hat{I}_{p_{\text{tar}}}$, which we apply to align $I_{p_{\text{tar}}}$ and $\hat{I}_{p_{\text{tar}}}$ as well as $I_{p_{\text{src}} \to p_{\text{tar}}}$ via

$$I^{\mathsf{A}} \doteq \texttt{Align}(I, (\Delta h, \Delta w)), \tag{12}$$

where I^{A} marks aligned images. We then use the three aligned images as input:

$$\begin{aligned} \mathcal{L} =& \lambda \| I^{\mathsf{A}}_{p_{\text{tar}}} - \hat{I}^{\mathsf{A}}_{p_{\text{tar}}} \|_1 + \mathbb{E}_{I^{\mathsf{A}}_{p_{\text{tar}}}, I^{\mathsf{A}}_{p_{\text{src}} \to p_{\text{tar}}}} \left[\log D(I^{\mathsf{A}}_{p_{\text{src}} \to p_{\text{tar}}} | I^{\mathsf{A}}_{p_{\text{tar}}}) \right] \\ &+ \mathbb{E}_{I^{\mathsf{A}}_{p_{\text{tar}}}, \hat{I}^{\mathsf{A}}_{p_{\text{tar}}}} \left[\log(1 - D(\hat{I}^{\mathsf{A}}_{p_{\text{tar}}} | I^{\mathsf{A}}_{p_{\text{tar}}})) \right]. \end{aligned} \tag{13}$$

Here, D is a convolutional deep-net based discriminator. When using the unaligned image I instead of I^{A}, Eq. (13) reduces to the vanilla version in [24]. We now discuss a fast way to align images.

Alignment with Fourier Transformation: To align ground truth $I_{p_{\text{tar}}}$ and rendering $\hat{I}_{p_{\text{tar}}}$, one could use naïve grid-search to find the offset which results in the minimum difference of the shifted images. However, such a grid-search is prohibitively costly during an iterative optimization, especially with high-resolution images (*e.g.*, we use a resolution up to 1920×1440). Instead, we use the fast Fourier transformation (FFT) to complete the job [6]. Specifically, given a misaligned image pair of $I \in \mathbb{R}^{h \times w \times 3}$ and $\hat{I} \in \mathbb{R}^{h \times w \times 3}$, we compute for every channel the maximum correlation via

[1] https://github.com/hjwdzh/AdversarialTexture.

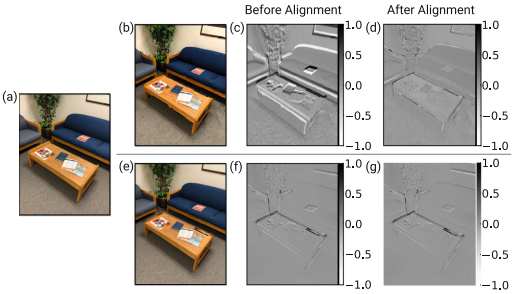


Fig. 7. Alignment with fast Fourier transformation (FFT) (Sect. 3.2). We show results for $\mathcal{T}_{\text{init}}$ (Sect. 3.1) and L2Avg (Fig. 5) in top and bottom rows. **(a):** ground-truth (GT); **(b) and (e):** texture rendering with (a)'s corresponding camera; **(c) and (d):** difference between (a) and (b); **(f) and (g):** difference between (a) and (e). The top row: FFT successfully aligns GT image and rendering from $\mathcal{T}_{\text{init}}$. Within expectation, there is almost no misalignment for the texture L2Avg as it overfits to available views (Fig. 5).

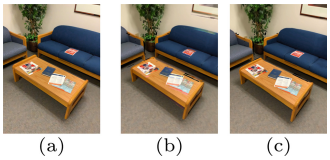


Fig. 8. Alignment is important for evaluation (Sect. 4.1). Clearly, (c) is more desirable than (b). However, before alignment, LPIPS yields 0.3347 and 0.4971 for (a)–(b) and (a)–(c) pairs respectively. This is misleading as lower LPIPS indicates higher quality. After alignment, LPIPS produces 0.3343 and 0.2428 for the same pairs, which provides correct signals for evaluation.

$$\underset{(i,j)}{\text{argmax}}\ \mathsf{FFT}^{-1}\left(\mathsf{FFT}(I)\cdot\overline{\mathsf{FFT}(\hat{I})}\right)[i,j]. \tag{14}$$

Here, $\mathsf{FFT}(\cdot)$ represents the fast Fourier transformation while $\mathsf{FFT}^{-1}(\cdot)$ denotes its inverse and $\overline{\mathsf{FFT}(\hat{I})}$ refers to the complex conjugate. After decoding the maximum correlation response and averaging across channels, we obtain the final offset $(\Delta h, \Delta w)$. As can be seen in Fig. 7(d), the offset $(\Delta h, \Delta w)$ is very accurate. Moreover, the computation finishes in around 0.4 s even for 1920×1440-resolution images. Note, we don't need to maintain gradients for $(\Delta h, \Delta w)$, since the offset is only used to shift images and not to backpropagate through it.

4 Experiments

4.1 Experimental Setup

Data Acquisition. We use a 2020 iPad Pro and develop an iOS app to acquire the RGBD images I_t, camera pose p_t, and scene mesh M via Apple's ARKit [1].

UofI Texture Scenes. We collect a dataset of 11 scenes: four indoor and seven outdoor scenes. This dataset consists of a total of 2807 frames, of which 91, 2052, and 664 are of resolution 480 × 360, 960 × 720, and 1920 × 1440 respectively.

For each scene, we use 90% of its views for optimization and the remainder for evaluation. In total, we have 2521 training frames and 286 test frames. This setting is more challenging than prior work where [24] "select(s) 10 views uniformly distributed from the scanning video" for evaluation while using up to thousands of frames for texture generation. On average, the angular differences between test set view directions and their nearest neighbour in the training sets are 2.05° (min 0.85°/max 13.8°). Angular distances are computed following [25]. Please see Appendix for scene-level statistics.

Implementation. We compare to five baselines for texture generation: L2Avg, ColorMap [54], TexMap [16], MVSTex [48], and AdvTex [24]. For ColorMap, TexMap, and MVSTex, we use their official implementations.[2] For `AdvOptim` (Sect. 3.2) used in both AdvTex and ours, we re-implement a PyTorch [35] version based on their official release in TensorFlow [2] (See footnote 1). We evaluate AdvTex with two different initializations: 1) blank textures as stated in the paper (AdvTex-B); 2) the initialization used in the official code release (AdvTex-C). We run `AdvOptim` using the Adam optimizer [26]. See Appendix for more details. For our `TexInit` (Sect. 3.1), we use a generic set of weights across all scenes: $\omega_1 = 1e^{-3}$ (triangle area), $\omega_2 = -10$ (depth discrepancy), and $\omega_3 = -1$ (perception consistency), which makes cue magnitudes roughly similar.

On a 3.10 GHz Intel Xeon Gold 6254 CPU, ColorMap takes less than two minutes to complete while TexMap's running time ranges from 40 min to 4 h. MVSTex can be completed in no more than 10 min. Our $\mathcal{T}_{\text{init}}$ (Sect. 3.1) completes in two minutes. Additionally, the `AdvOptim` takes around 20 min for 4000 iterations to complete with an Nvidia RTX A6000 GPU.

Evaluation Metrics. To assess the efficacy of the method, we study the quality of the texture from two perspectives: perceptual quality and sharpness. 1) For perceptual quality, we assess the similarity between rendered and collected ground-truth views using the Structural Similarity Index Measure (SSIM) [49] and the Learned Perceptual Image Patch Similarity (LPIPS) [53]. 2) For sharpness, we consider measurement S_3 [47] and the norm of image gradient (Grad) following [24]. Specifically, for each pixel, we compute its S_3 value, whose difference between the rendered and ground truth (GT) is used for averaging across the whole image. A similar procedure is applied to Grad. For all four metrics, we report the mean and standard deviation across 11 scenes.

Alignment in Evaluation. As can be seen in Fig. 8, evaluation will be misleading if we do not align images during evaluation. Therefore, we propose the following procedure: 1) for each method, we align the rendered image and the GT using an `FFT` (Sect. 3.2); 2) to avoid various resolutions caused by different methods, we crop out the maximum common area across methods. 3) we then

[2] ColorMap: https://github.com/intel-isl/Open3D/pull/339; TexMap: https://github.com/fdp0525/G2LTex; MVSTex: https://github.com/nmoehrle/mvs-texturing.

compute metrics on those cropped regions. The resulting comparison is fair as all methods are evaluated on the same number of pixels and aligned content.

4.2 Experimental Evaluation

Quantitative Evaluation. Table 1 reports aggregated results on all 11 scenes. The quality of our texture $\mathcal{T}$ (Row 6) outperforms baselines on LPIPS, S_3 and Grad, confirming the effectiveness of the proposed pipeline. Specifically, we improve LPIPS by 7.8% from 0.335 (2^{nd}-best) to 0.309, indicating high perceptual similarity. Moreover, $\mathcal{T}$ maintains sharpness as we improve S_3 by 11.1% from 0.135 (2^{nd}-best) to 0.120 and Grad from 7.171 (2^{nd}-best) to 6.871. Regarding SSIM, we find it to favor L2Avg in almost all scenes (see Appendix) which aligns with the findings in [53].

Table 1. Aggregated quantitative evaluation on UofI Texture Scenes. We report results in the form of `mean`±`std`. Please see Fig. 9 for qualitative texture comparisons and Appendix for scene-level quantitative results.

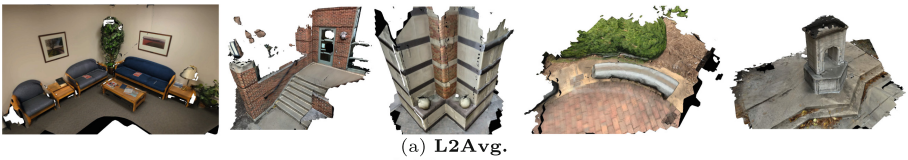
		SSIM↑	LPIPS↓	S_3 ↓	Grad↓
1	L2Avg	**0.610** ± 0.191	0.386 ± 0.116	0.173 ± 0.105	7.066 ± 4.575
2	ColorMap	0.553 ± 0.193	0.581 ± 0.132	0.234 ± 0.140	7.969 ± 5.114
3	TexMap	0.376 ± 0.113	0.488 ± 0.097	0.179 ± 0.062	8.918 ± 4.174
4	MVSTex	0.476 ± 0.164	0.335 ± 0.086	0.139 ± 0.047	8.198 ± 3.936
5-1	AdvTex-B	0.495 ± 0.174	0.369 ± 0.092	0.148 ± 0.047	8.229 ± 4.586
5-2	AdvTex-C	0.563 ± 0.191	0.365 ± 0.096	0.135 ± 0.067	7.171 ± 4.272
6	Ours	0.602 ± 0.189	**0.309** ± 0.086	**0.120** ± 0.058	**6.871** ± 4.342

Table 2. Ablation study. We report results in the form of `mean`±`std`.

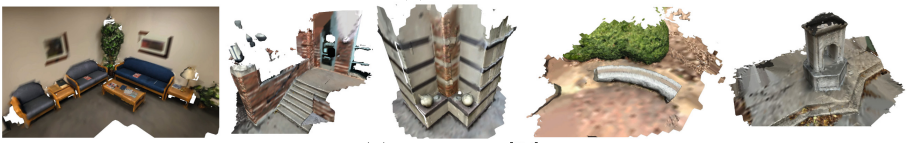
	Adv Optim	FFT Align	$\mathcal{T}_{\text{init}}$	SSIM↑	LPIPS↓	S_3 ↓	Grad↓
1			✓	0.510 ± 0.175	0.342 ± 0.060	0.141 ± 0.052	8.092 ± 4.488
2	✓	✓		0.592 ± 0.192	0.332 ± 0.102	0.130 ± 0.066	**6.864** ± 4.211
3	✓		✓	0.559 ± 0.196	0.346 ± 0.082	0.125 ± 0.057	7.244 ± 4.359
4	✓	✓	✓	**0.602** ± 0.189	**0.309** ± 0.086	**0.120** ± 0.058	6.871 ± 4.342

Ablation Study. We verify the design choices of `TexInit` and `TexSmooth` in Table 2. **1) `TexSmooth` is required:** we directly evaluate $\mathcal{T}_{\text{init}}$ and 1^{st} *vs.* 4^{th} row confirms the performance drop: −0.092 (SSIM), +0.033 (LPIPS), +0.021 (S_3), and +1.221 (Grad). **2) $\mathcal{T}_{\text{init}}$ is needed:** we replace $\mathcal{T}_{\text{init}}$ with L2Avg as it performs better than ColorMap and TexMap in Table 1 and still incorporate `FFT` into `AdvOptim`. We observe inferior performance: −0.010 (SSIM), +0.023 (LPIPS), +0.010 (S_3) in 2^{nd} *vs.* 4^{th} row. **3) Alignment is important:** we use the vanilla `AdvOptim` but initialize with $\mathcal{T}_{\text{init}}$. As shown in Table 2's 3^{rd} *vs.* 4^{th} row, the texture quality drops by −0.043 (SSIM), +0.037 (LPIPS), +0.005 (S_3), and +0.373 (Grad).

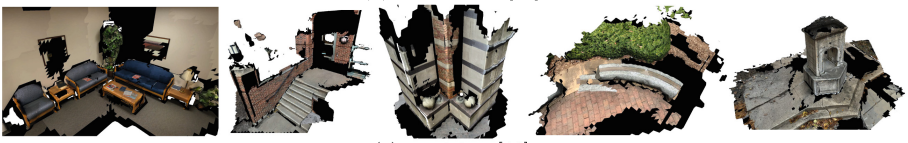
Qualitative Evaluation. We present qualitative examples in Fig. 9. Figure 9a and Fig. 9b demonstrate that L2Avg and ColorMap produce overly smooth texture. Meanwhile, due to noise in the geometry, *e.g.*, Fig. 2, TexMap fails to



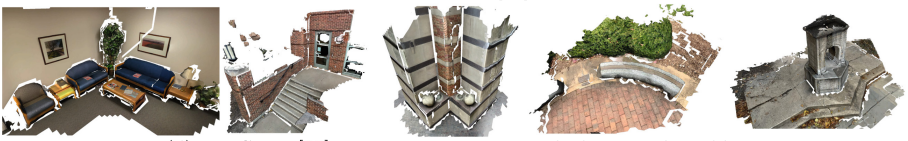
(a) **L2Avg.**



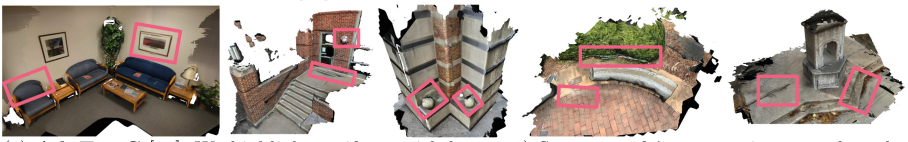
(b) **ColorMap** [54].



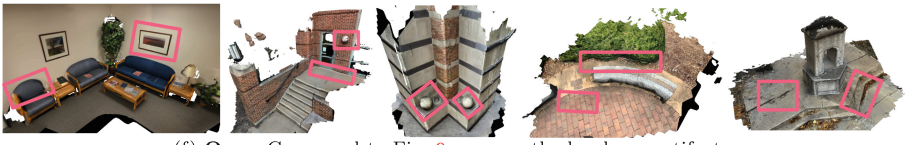
(c) **TexMap** [16].



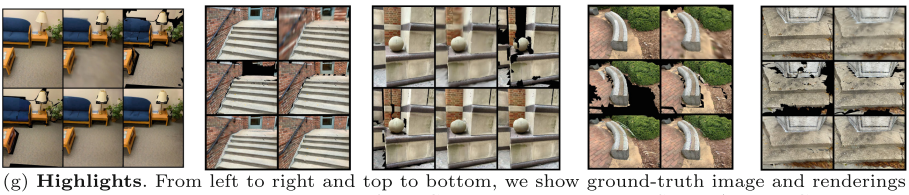
(d) **MVSTex** [48]. It removes geometry, which is not desirable.



(e) **AdvTex-C** [24]. We highlight artifacts with boxes. 1) Scene 1: sofa's texture is mapped to the wall and the figure on the wall is broken; 2) Scene 2: the door's color is mapped to the floor and the brick wall's pattern is mapped to light; 3) Scene 3: ball's color is projected to brick walls; 4) Scene 4: bench's color is added to the bush and ground; 5) Scene 5: the crack breaks and stair's color is on the ground.



(f) **Ours**. Compared to Fig. 9e, our method reduces artifacts.



(g) **Highlights**. From left to right and top to bottom, we show ground-truth image and renderings at the same camera pose with texture from ColorMap, TexMap, MVSTex, AdvTex-C, and ours respectively. Compared to AdvTex-C: 1) Scene 1: ours produces much more complete and sharper pattern for the sofa; 2) Scene 2: ours generates sharper cracks on brick walls; 3) Scene 3: our balls are more complete while AdvTex-C maps the top of the ball to the left brick wall; 4) Scene 4: AdvTex-C maps the bench to the bush while ours is much cleaner; 5) Scene 5: the pattern on the ground from ours is much sharper.

Fig. 9. Qualitative results on UofI Texture Scenes. For each method, we show results for Scene 1 to 5 from left to right. Best viewed in color and zoomed-in. (Color figure online)

Table 3. ScanNet results. We report results in the form of mean±std. Note, this can't be directly compared to [23]'s Table 2: while we reserve 10% views for evaluation, [23] reserves only 10/2011 (≈0.5%), where 2011 is the number of average views per scene.

		SSIM↑	LPIPS↓	S_3 ↓	Grad↓
1-1	AdvTex-B	0.534 ± 0.074	0.557 ± 0.071	0.143 ± 0.028	3.753 ± 0.730
1-2	AdvTex-C	0.531 ± 0.074	0.558 ± 0.075	0.161 ± 0.044	4.565 ± 1.399
2	Ours	**0.571** ± 0.069	**0.503** ± 0.090	**0.127** ± 0.031	**3.324** ± 0.826

Fig. 10. Remaining six scenes with our textures. See Appendix for all methods' results on these scenes.

Fig. 11. Results on ScanNet. Left to right: AdvTex-B/C and ours. Ours alleviates artifacts: colors from box on the cabinet are mapped to the backpack and wall.

resolve texture seams and cannot produce a complete texture (Fig. 9c). MVSTex results in Fig. 9d are undesirable as geometries are removed. This is because MVSTex requires ray collision checking to remove occluded faces. Due to the misalignment between geometries and cameras, artifacts are introduced. We show results of AdvTex-C in Fig. 9e as it outperforms AdvTex-B from Table 1. Artifacts are highlighted. Our method can largely mitigate such seams, which can be inferred from Fig. 9f. In Fig. 9g, we show renderings, which demonstrate the effectiveness of the proposed method. Please see Appendix for complete qualitative results of scenes in Fig. 10.

On ScanNet [11]**.** Following [24], we study scenes with ID ≤ 20 (Fig. 11, Table 3). We improve upon baselines (AdvTex-B/C) by a margin on SSIM ($0.534 \rightarrow 0.571$), LPIPS ($0.557 \rightarrow 0.503$), S_3 ($0.143 \rightarrow 0.127$), and Grad ($3.753 \rightarrow 3.324$).

5 Conclusion

We develop an initialization and an alignment method for fully-automatic texture generation from a given scene mesh, and a given sequence of RGBD images and their camera parameters. We observe the proposed method to yield appealing results, addressing robustness issues due to noisy geometry and misalignment of prior work. Quantitatively we observe improvements on both perceptual similarity (LPIPS from 0.335 to 0.309) and sharpness (S_3 from 0.135 to 0.120).

Acknowledgements. Supported in part by NSF grants 1718221, 2008387, 2045586, 2106825, MRI #1725729, and NIFA award 2020-67021-32799.

References

1. Augmented Reality - Apple Developer. https://developer.apple.com/augmented-reality/ (2021). Accessed 14 Nov 2021
2. Abadi, M., et al: TensorFlow: a system for large-scale machine learning. In: OSDI (2016)
3. Abdelhafiz, A., Mostafa, Y.G.: Automatic texture mapping mega-projects. J. Spatial Sci. (2020)
4. Aganj, E., Monasse, P., Keriven, R.: Multi-view texturing of imprecise mesh. In: Zha, H., Taniguchi, R., Maybank, S. (eds.) ACCV 2009. LNCS, vol. 5995, pp. 468–476. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12304-7_44
5. Aliev, K.-A., Sevastopolsky, A., Kolos, M., Ulyanov, D., Lempitsky, V.: Neural point-based graphics. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12367, pp. 696–712. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58542-6_42
6. Anuta, P.E.: Spatial registration of multispectral and multitemporal digital imagery using fast Fourier Transform techniques. IEEE Trans. Geosci. Electron. **8**(4), 353–368 (1970)
7. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: PatchMatch: a randomized correspondence algorithm for structural image editing. In: SIGGRAPH (2009)
8. Baumberg, A.: Blending images for texturing 3D models. In: BMVC (2002)
9. Bernardini, F., Martin, I., Rushmeier, H.: High quality texture reconstruction from multiple scans. TVCG **7**(4), 318–332 (2001)
10. Bi, S., Kalantari, N.K., Ramamoorthi, R.: Patch-based optimization for image-based texture mapping. TOG **36**(4), 106-1 (2017)
11. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T.A., Nießner, M.: ScanNet: richly-annotated 3D reconstructions of indoor scenes. In: CVPR (2017)
12. Debevec, P., Taylor, C., Malik, J.: Modeling and rendering architecture from photographs: a hybrid geometry and image-based approach. In: SIGGRAPH (1996)
13. Duan, Y.: Topology adaptive deformable models for visual computing. Ph.D. thesis, State University of New York (2003)
14. El-Hakim, S., Gonzo, L., Picard, M., Girardi, S., Simoni, A.: Visualization of Frescoed surfaces: Buonconsiglio Castle - Aquila Tower, "Cycle of the Months". IAPRS (2003)
15. Früh, C., Sammon, R., Zakhor, A.: Automated texture mapping of 3D city models with oblique aerial imagery. In: 3DPVT (2004)
16. Fu, Y., Yan, Q., Yang, L., Liao, J., Xiao, C.: Texture mapping for 3D reconstruction with RGB-D sensor. In: CVPR (2018)
17. Globerson, A., Jaakkola, T.: Fixing max-product: convergent message passing algorithms for MAP LP-relaxations. In: NIPS (2007)
18. Goel, S., Kanazawa, A., Malik, J.: Shape and viewpoint without keypoints. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12360, pp. 88–104. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58555-6_6

19. Grammatikopoulos, L., Kalisperakis, I., Karras, G., Petsa, E.: Automatic multi-view texture mapping of 3D surface projections. In: International Workshop 3D-ARCH (2007)
20. Groueix, T., Fisher, M., Kim, V.G., Russell, B.C., Aubry, M.: AtlasNet: a Papier-Mâché approach to learning 3D surface generation. In: CVPR (2018)
21. Henzler, P., Mitra, N.J., Ritschel, T.: Learning a neural 3D texture space from 2D exemplars. In: CVPR (2020)
22. Hernández-Esteban, C.: Stereo and Silhouette fusion for 3D object modeling from uncalibrated images under circular motion. Ph.D. thesis, École Nationale Supérieure des Télécommunications (2004)
23. Huang, J., Dai, A., Guibas, L., Nießner, M.: 3DLite: towards commodity 3D scanning for content creation. ACM TOG **36**, 1–14 (2017)
24. Huang, J., et al.: Adversarial texture optimization from RGB-D scans. In: CVPR (2020)
25. Huynh, D.: Metrics for 3D rotations: comparison and analysis. J. Math. Imaging Vision **35**, 155–164 (2009)
26. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. ArXiv (2015)
27. Lempitsky, V., Ivanov, D.: Seamless Mosaicing of image-based texture maps. In: CVPR (2007)
28. Lensch, H., Heidrich, W., Seidel, H.P.: Automated texture registration and stitching for real world models. In: Graphical Models (2001)
29. Marshner, S.R.: Inverse rendering for computer graphics. Ph.D. thesis, Cornell University (1998)
30. Neugebauer, P.J., Klein, K.: Texturing 3D models of real world objects from multiple unregistered photographic views. In: Eurographics (1999)
31. Niem, W., Wingbermühle, J.: Automatic reconstruction of 3D objects using a mobile camera. In: IVC (1999)
32. Oechsle, M., Mescheder, L.M., Niemeyer, M., Strauss, T., Geiger, A.: Texture fields: learning texture representations in function space. In: ICCV (2019)
33. Ofek, E., Shilat, E., Rappoport, A., Werman, M.: Multiresolution textures from image sequences. Comput. Graph. Appl. **17**(2), 18–29 (1997)
34. Pan, R., Taubin, G.: Color adjustment in image-based texture maps. Graph. Models **79**, 39–48 (2015)
35. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. ArXiv abs/1912.01703 (2019)
36. Pighin, F., Hecker, J., Lischinski, D., Szeliski, R., Salesin, D.H.: Synthesizing realistic facial expressions from photographs. In: CGIT (1998)
37. Ravi, N., et al.: Accelerating 3D deep learning with PyTorch3D. arXiv:2007.08501 (2020)
38. Rocchini, C., Cignoni, P., Montani, C., Scopigno, R.: Multiple textures stitching and blending on 3D objects. In: Eurographics Workshop on Rendering (1999)
39. Rocchini, C., Cignoni, P., Montani, C., Scopigno, R.: Acquiring, stitching and blending diffuse appearance attributes on 3D models. Visual Comput. **18**, 186–204 (2002)
40. Saito, S., Wei, L., Hu, L., Nagano, K., Li, H.: Photorealistic facial texture inference using deep neural networks. In: CVPR (2017)
41. Sawhney, R., Crane, K.: Boundary first flattening. ACM TOG **37**, 1–14 (2018)
42. Shu, J., Liu, Y., Li, J., Xu, Z., Du, S.: Rich and seamless texture mapping to 3D mesh models. In: Tan, T., et al. (eds.) IGTA 2016. CCIS, vol. 634, pp. 69–76. Springer, Singapore (2016). https://doi.org/10.1007/978-981-10-2260-9_9

43. Sinha, S.N., Steedly, D., Szeliski, R., Agrawala, M., Pollefeys, M.: Interactive 3D architectural modeling from unordered photo collections. In: SIGGRAPH 2008 (2008)
44. Sitzmann, V., Thies, J., Heide, F., Nießner, M., Wetzstein, G., Zollhöfer, M.: DeepVoxels: learning persistent 3D feature embeddings. In: CVPR (2019)
45. Thierry, M., David, F., Gorria, P., Salvi, J.: Automatic texture mapping on real 3D model. In: CVPR (2007)
46. Thies, J., Zollhöfer, M., Nießner, M.: Deferred neural rendering. ACM TOG **38**, 1–12 (2019)
47. Vu, C.T., Phan, T.D., Chandler, D.M.: S_3: a spectral and spatial measure of local perceived sharpness in natural images. IEEE Trans. Image Process. **21**(3), 934–945 (2012)
48. Waechter, M., Moehrle, N., Goesele, M.: Let there be color! Large-scale texturing of 3D reconstructions. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 836–850. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_54
49. Wang, Z., Bovik, A., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)
50. Wexler, Y., Shechtman, E., Irani, M.: Space-time video completion. In: CVPR (2004)
51. Wuhrer, S., Atanassov, R., Shu, C.: Fully automatic texture mapping for image-based modeling. Technical report, Institute for Information Technology (2006)
52. Xiang, F., Xu, Z., Havsan, M., Hold-Geoffroy, Y., Sunkavalli, K., Su, H.: NeuTex: neural texture mapping for volumetric neural rendering. In: CVPR (2021)
53. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: CVPR (2018)
54. Zhou, Q.Y., Koltun, V.: Color map optimization for 3D reconstruction with consumer depth cameras. ACM TOG **33**(4), 1–10 (2014)