EUROGRAPHICS Workshop on Graphics and Cultural Heritage (2024)
M. Corsini, D. Ferdani, A. Kuijper, and H. Kutlu (Editors)

Short Paper

DeepTex: Deep Learning-Based Texturing of Image-Based 3D Reconstructions

K. A. Neumann[1] and P. Santos[1] and D. W. Fellner[1,2,3]

[1] Fraunhofer Institute for Computer Graphics Research IGD, Germany
[2]TU Darmstadt, Germany
[3]Graz University of Technology, Institute of Computer Graphics and Knowledge Visualization, Austria

Abstract

Image-based 3D reconstruction is a commonly used technique for measuring the geometry and color of objects or scenes based on images. While the geometry reconstruction of state-of-the-art approaches is mostly robust against varying lighting conditions and outliers, these pose a significant challenge for calculating an accurate texture map. This work proposes a deep-learning based texturing approach called "DeepTex" that uses a custom learned blending method on top of a traditional mosaic-based texturing approach. The model was trained using a custom synthetic data generation workflow and showed a significantly increased accuracy when generating textures in the presence of outliers and non-uniform lighting.

CCS Concepts

• ***Computing methodologies*** → ***Reconstruction;*** *Computer vision;* ***Artificial intelligence;***

1. Introduction

Image-based 3D reconstruction is a widely utilized technique that aims to infer the geometry and color of objects or scenes from corresponding images. Its applications range from the preservation of artifacts for cultural heritage to quality control in manufacturing processes.

One significant challenge in image-based 3D reconstructions is accurately calculating the color for each point on an object's surface, a process commonly referred to as "texturing". This is particularly challenging when the input images contain varying lighting conditions or outliers. For instance, if a building is being 3D-reconstructed based on tourist images taken from the internet, many images will contain outliers, such as pedestrians, that should not be included in the final texture. Similar issues may arise when objects are highly reflective and exhibit specular highlights that vary based on the camera position during capture. These specular highlights should also be excluded from the calculated texture.

While numerous previous approaches for robust texturing have been proposed, they frequently fail to completely remove visual artifacts for challenging inputs. For this reason, a novel deep learning-based approach is presented in this article that can accurately and robustly calculate textures in the presence of outliers, specular highlights, and varying lighting conditions. Our contributions can be summarized as:

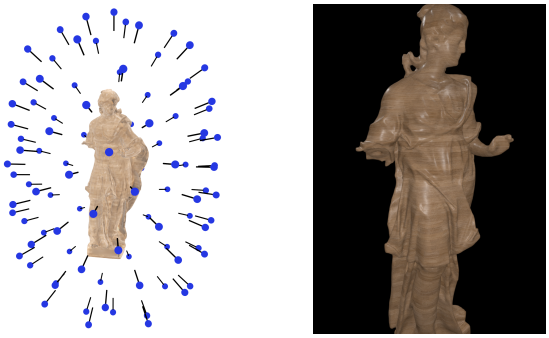
1. A novel deep learning-based texturing approach that shows improved robustness against outliers and varying lighting conditions.
2. A simulation framework for generating synthetic data for training and evaluating texturing algorithms.
3. A systematic comparison between our approach and existing approaches.

2. Related Work

Texturing approaches can generally be divided into blending-based and mosaic-based approaches. Blending-based approaches, calculate the color of a point as a weighted combination of all images that observe the point. For example, FRAHM et al. [FFG*10] propose calculating the color of a point as the average over all observed colors. In comparison, CALLIERI et al. [CCCS08] utilize a weighted blending factoring in the view angle, depth, model borders and focus distance. BAUMBERG [Bau02] proposes an alternative approach based on multi-band blending, which first splits the input images into low and high frequencies and blends them separately.

Mosaic-based texturing approaches assign a single image to each point or triangle and optionally perform additional post-processing steps to remove visible seams and artifacts. For example, after assigning an image to each triangle, ALLENE et al. [APK08] apply a pixel-wise color correction based on multi-band blending. LEMPITSKY et al. [LI07] propose solving the image assignment in form of a *Markov random field (MRF)* that optimizes two energy terms corresponding to the quality of the corresponding image and the smoothness of the resulting texture. The authors also propose the simultaneous optimization of a color adjustment map that

DOI: 10.2312/gch.20241257



(a) *Calculated poses.* **(b)** *Rendered image.*

Figure 1: *Synthetic data generation. The poses (blue) are placed in a spherical fashion around the object, based on its bounding box. The image is rendered in real time using the Godot engine [God24].*

is multiplied with the assigned colors to produce smooth textures. GAL et al. [GWO*10] propose an extension of the method by LEMPITSKY et al. that allows for translation of input images to overcome artifacts due to misalignment. WAECHTER et al. [WMG14] further extends the method of LEMPITSKY et al. by introducing a modified smoothness energy term, a photo consistency check to detect outliers and an improved global color adjustment.

Recent developments introduce AI-based approaches to texturing. XIANG et al. [XXH*21] introduce an editable texture mapping for *neural radiance fields (NeRF)*. Additionally, DENG et al. [DOW*24] and YEH et al. [YHK*24] propose generative approaches for texturing a model based on an input text prompt. However, none of these deep learning-based methods are applicable to texturing 3D models based on captured images.

3. Methods

3.1. Synthetic Data Generation

Training of deep learning-based approaches usually requires a large amount of data. Because of this, a custom rendering-based synthetic data generator was implemented to generate the necessary input data for the training and evaluation of the presented texturing workflow. The data generator was implemented using the *Godot Engine* [God24] leveraging its real time renderer to produce thousands of images in a matter of minutes.

The input required for texturing an object consists of three parts: a set of images, a set of corresponding camera poses, and a 3D model of the object. Given a 3D model and an arbitrary ground truth texture for that model the presented synthetic data generator can produce a set of images with corresponding poses (see Figure 1), that can be used for recovering the texture of the input 3D model. During generation, the camera poses are dynamically calculated based on the bounding box of the input 3D model.

The complete training dataset is generated from a set of 14 models from the cultural heritage domain [Neu24] as well as a set of 32 materials from *ambientCG* [Dem24]. Each material consists of an albedo map, a normal map, and a roughness map allowing the use of physically based rendering (PBR) in Godot. A selection of in total 512 random object-material combinations was used to produce our training dataset. All source data, source code, and the generated datasets were published at `https://github.com/kai-neumann/DeepTex`.

Rendering-based data generation allows for easy variation of lighting conditions as well as disturbing the rendered images to introduce outliers into the texturing datasets. In the presented workflow, three different image sets are generated for each object-material combination. First, an image set with perfectly uniform ambient lighting, which is used to generate a ground truth texture. Second, an image set where the model is illuminated by a light source placed directly behind the camera. This mimics capture scenarios where images are captured using a camera flash. The third image set uses the same lighting conditions as the second, but includes a random disturbance per image. This includes varying the brightness and contrast of the image, blurring the image, and applying a small random rotation or translation to the camera that does not match the saved camera pose.

3.2. Texture Projection

The first stage of the presented deep learning-based texturing pipeline is based on a classical mosaic-based texturing approach (see [APK08; LI07; WMG14]). This step produces a set of candidate colors with assigned quality scores per pixel which we use as input for our subsequent deep learning-based blending approach.

The first step for calculating candidate colors consists of projecting the corresponding 3D point of each pixel onto all images in which it is visible. Here, the corresponding 3D point $\mathbf{x}_i \in \mathbb{R}^3$ of a pixel with normalized UV-coordinates (u_i, v_i) is calculated by identifying the triangle in UV-space that contains (u_i, v_i) using barycentric coordinates. The same barycentric coordinates can then be reused to calculate the 3D coordinates of the corresponding 3D point $\mathbf{x}_i$ based on the 3D coordinates of the triangles vertices.

After calculating the corresponding 3D point $\mathbf{x}_i \in \mathbb{R}^3$, the next step consists of performing visibility checks to identify in what images the point is visible. This is possible by casting a ray from the current point to the origin of each camera. To do this efficiently, a *bounding volume hierarchy (BVH)* is used as acceleration structure. If a ray intersects with the model on its way between the surface point and camera origin, the corresponding surface point is occluded. Subsequently, only the non-occluded cameras are used to calculate the candidate colors of each pixel by looking up the color of the 3D points projection onto the respective image plane.

In addition to the RGB values of each candidate color, a score is calculated per candidate color to quantify how well the corresponding surface point was captured by the respective camera. Similar to the approach of CALLIERI et al. [CCCS08], this is done by factoring in the viewing angle and the distance of the given point to the focus plane. Before continuing with the deep learning-based blending, the candidate colors of each pixel are sorted in decreasing order based on their score. A first texture can be obtained at this point by greedily assigning the color with the highest score to each pixel. This greedy texturing algorithm was included as a baseline in the evaluation (see Section 4).

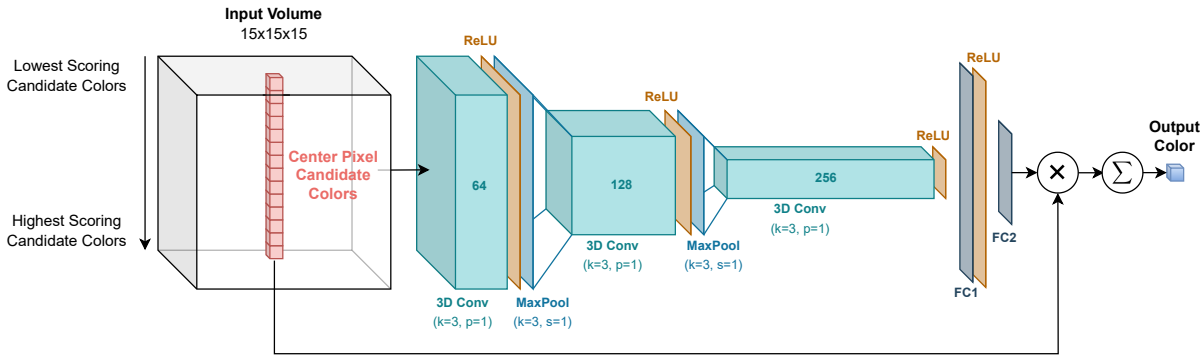


Figure 2: *Overview of the proposed network structure. The input volume is processed using three consecutive 3D convolution blocks (3D Conv), each followed by a ReLU activation and a MaxPooling operation (MaxPool). Afterwards two fully connected layers (FC) are used to produce weight vector that is multiplied with the 15 candidate colors of the center pixel to produce and output color.*

3.3. Deep Learning-based Blending

The deep learning-based blending utilizes a *3D convolutional neural network (CNN)* that is executed per pixel to produce an output color. As an input, this network considers a (15×15) pixel neighborhood around each pixel taking into account the 15 highest scoring candidate colors that were calculated using the previously introduced texture projection. This results in a total input size of $(15 \times 15 \times 15 \times 3)$ values per resulting pixel. In practice, it is possible for a pixel to have less than 15 candidate colors. In this case the input data is padded with zero values.

The input volume is processed using three consecutive 3D convolution blocks, each followed by a ReLU activation and a MaxPooling operation (see Figure 2). The output of the last convolution is used as an input for two fully connected layers with the second layer producing exactly 45 values as an output. These values are used to calculate a channel-wise weighted product of the 15 candidate colors of the current (center) pixel resulting in a single output color. This part is critical, as it forces the 3D-CNN to learn a set of weights for the input colors instead of directly learning a mapping from input colors to output colors. In our tests this led to an improved ability to generalize as well overall sharper textures, as oversmoothing is significantly more difficult with this network design.

For training, a combination of two losses was used. First, a *mean squared error (MSE)* $\mathcal{L}_{\text{mse}}$ was calculated to measure the distance between the predicted color of the current pixel and the corresponding ground truth color. To ensure that the predicted color is also consistent to its neighborhood a second loss $\mathcal{L}_{\text{ssim}}$ was introduced that is based on the *structural similarity index measure (SSIM)*[WBSS04]. Here, the ground truth values of the 15×15 neighborhood are augmented by replacing the center pixel with the predicted color. Calculating an SSIM between the modified and unmodified neighborhood quantifies how well the predicted color fits into its neighborhood, producing a value of $s = 1.0$ if the pixel fits perfectly and a value close to zero if it is completely incoherent with its neighborhood. The corresponding loss is defined as $\mathcal{L}_{\text{ssim}} = 1 - s$, resulting in the combined loss

$$\mathcal{L} = \frac{1}{4}\left(\mathcal{L}_{\text{mse}} + 3\mathcal{L}_{\text{ssim}}\right). \tag{1}$$

Here, the SSIM-loss is weighted higher to increase the importance of a pixels coherence with its neighborhood. The model was trained over 45 epochs using the *Adam Optimizer* [KB14] with an initial learning rate of $\alpha = 0.0001$ and an exponential learning rate scheduler that reduced the learning rate by a factor of 10 every 10 epochs.

4. Results

The presented synthetic data generation was used to create image sets from 128 random model-material combinations to evaluate the accuracy of our proposed texturing workflow in comparison to existing algorithms. Each image set was augmented with 25% noisy images to provide numerous outliers. Specifically, a greedy mosaic-based texturing, the state-of-the-art MRF-based blending approach *MVS-Texturing* [WMG14] and the commercial software *Agisoft Metashape* were used for benchmarking. All textures were set to be generated at a resolution of 1024×1024 pixels.

The resulting textures from all algorithms were compared to the ground truth textures using both the mean squared error (MSE) and the multiscale SSIM (see 1). Notably, both metrics were modified to include only pixels that are part of the mesh to avoid biasing the results based on unfilled texture space.

According to the quantitative results, the proposed algorithm performs significantly better in terms of accuracy, producing a nearly optimal SSIM value. However, the runtime is approximately twofold that of the state-of-the-art algorithm, MVS-Texturing, and nearly eight times that of the baseline greedy algorithm. Nevertheless, the substantial increase in accuracy justifies this increased runtime. This is even more apparent when visually comparing the resulting textures (see 3). Compared to all other approaches the presented approach is the only one that is able to effectively remove non-uniform lighting and outliers.

	No Outlier Images						25% Outlier Images					
	SSIM		MSE		Runtime [s]		SSIM		MSE		Runtime [s]	
	Mean	*SD*	Mean	*SD*	Mean	*SD*	Mean	*SD*	Mean	*SD*	Mean	*SD*
Greedy Algorithm	0.9063	*0.0758*	0.004	*0.0027*	**8.3531**	*8.1196*	0.7868	*0.1038*	0.0119	*0.0061*	**8.3198**	*8.0852*
Agisoft Metashape	0.3737	*0.1736*	0.0063	*0.0053*	24.5319	*21.9542*	0.3537	*0.1628*	0.0079	*0.0062*	25.1723	*22.8336*
MVS-Texturing [WMG14]	0.7968	*0.1715*	0.0076	*0.0053*	34.8812	*38.2563*	0.7876	*0.1724*	0.0081	*0.0054*	35.9665	*39.1061*
DeepTex (ours)	**0.9786**	*0.0141*	**0.0005**	*0.0003*	65.2687	*14.5862*	**0.9608**	*0.0232*	**0.0009**	*0.0006*	65.1384	*14.0693*

Table 1: *Quantitative analysis of the texture accuracy and runtime when calculating a texture using multiple different approaches. The runtime was measured on a system with an AMD Ryzen 7 (3.60 GHz) and a NVIDIA GeForce RTX 4060 Laptop GPU.*

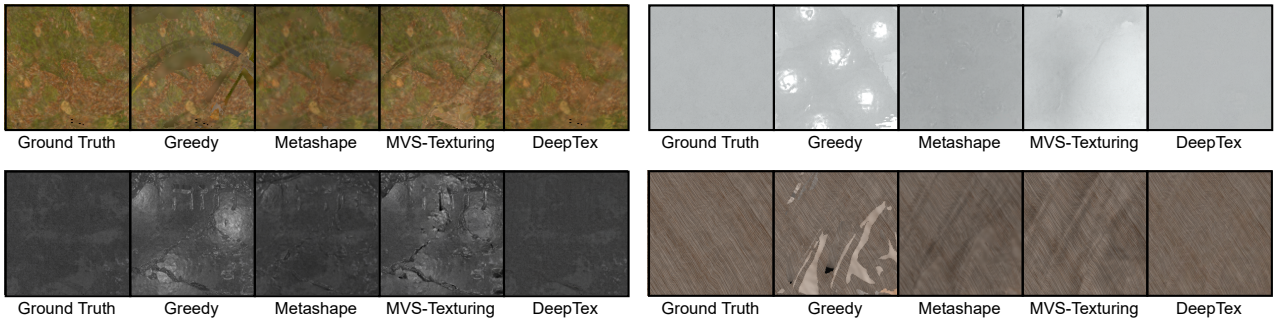


Figure 3: *Visual comparison of textures generated with the tested approaches. The textures were cropped to highlight relevant areas.*

5. Future Work

The most important goal for the future is to extensively test the approach on real world reconstruction data. Furthermore, the authors would like to explore the use of a more lightweight alternative to 3D-CNNs as well as improving runtime performance by performing model quantization.

6. Conclusion

This work introduces a novel deep learning-based texturing approach called "DeepTex" that shows and improved robustness against outliers and varying lighting conditions. The presented algorithm is based on a traditional mosaic-based texturing approach utilizing a 3D-CNN for robustly weighting observed colors. It has been trained using a custom synthetic data generation workflow that was also used for an evaluation and comparison to the state of the art.

References

[APK08] ALLENE, CÉDRIC, PONS, JEAN-PHILIPPE, and KERIVEN, RENAUD. "Seamless image-based texture atlases using multi-band blending". *2008 19th international conference on pattern recognition*. IEEE. 2008, 1–4 1, 2.

[Bau02] BAUMBERG, ADAM. "Blending Images for Texturing 3D Models." *Bmvc*. Vol. 3. 2002, 5 1.

[CCCS08] CALLIERI, MARCO, CIGNONI, PAOLO, CORSINI, MASSIMILIANO, and SCOPIGNO, ROBERTO. "Masked photo blending: Mapping dense photographic data set on high-resolution sampled 3D models". *Computers & Graphics* 32.4 (2008), 464–473 1, 2.

[Dem24] DEMES, LENNART. *ambientCG - CC0 Textures, HDRIs and Models*. https://ambientcg.com/. 2024 2.

[DOW*24] DENG, KANGLE, OMERNICK, TIMOTHY, WEISS, ALEXANDER, et al. "FlashTex: Fast Relightable Mesh Texturing with LightControlNet". *arXiv preprint arXiv:2402.13251* (2024) 2.

[FFG*10] FRAHM, JAN-MICHAEL, FITE-GEORGEL, PIERRE, GALLUP, DAVID, et al. "Building rome on a cloudless day". *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer. 2010, 368–381 1.

[God24] GODOT FOUNDATION. *Godot Engine - Free and open source 2D and 3D game engine*. https://godotengine.org/. 2024 2.

[GWO*10] GAL, RAN, WEXLER, YONATAN, OFEK, EYAL, et al. "Seamless montage for texturing models". *Computer Graphics Forum*. Vol. 29. 2. Wiley Online Library. 2010, 479–486 2.

[KB14] KINGMA, DIEDERIK P and BA, JIMMY. "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980* (2014) 3.

[LI07] LEMPITSKY, VICTOR and IVANOV, DENIS. "Seamless mosaicing of image-based texture maps". *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, 1–6 1, 2.

[Neu24] NEUMANN, KAI A. "Cultural Heritage Models for Benchmarking MVS Algorithms". (Apr. 2024). DOI: 10.6084/m9.figshare.25592400.v1. URL: https://figshare.com/articles/dataset/Cultural_Heritage_Models_for_Benchmarking_MVS_Algorithms/25592400 2.

[WBSS04] WANG, ZHOU, BOVIK, ALAN C, SHEIKH, HAMID R, and SIMONCELLI, EERO P. "Image quality assessment: from error visibility to structural similarity". *IEEE transactions on image processing* 13.4 (2004), 600–612 3.

[WMG14] WAECHTER, MICHAEL, MOEHRLE, NILS, and GOESELE, MICHAEL. "Let there be color! Large-scale texturing of 3D reconstructions". *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, 836–850 2–4.

[XXH*21] XIANG, FANBO, XU, ZEXIANG, HASAN, MILOS, et al. "Neutex: Neural texture mapping for volumetric neural rendering". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, 7119–7128 2.

[YHK*24] YEH, YU-YING, HUANG, JIA-BIN, KIM, CHANGIL, et al. "Texturedreamer: Image-guided texture synthesis through geometry-aware diffusion". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, 4304–4314 2.