Texture Atlas Compression Based on Repeated Content Removal

Yuzhe Luo
yzluo@zju.edu.cn
State Key Laboratory of CAD&CG,
Zhejiang University
LightSpeed Studios
China

Xiaogang Jin*
jin@cad.zju.edu.cn
State Key Laboratory of CAD&CG,
Zhejiang University
China

Zherong Pan
zrpan@global.tencent.com
LightSpeed Studios
USA

Kui Wu
kwwu@global.tencent.com
LightSpeed Studios
USA

Qilong Kou
kouqilong1988@gmail.com
Tencent Technology Co., LTD
China

Xiajun Yang
easonxjyang@tencent.com
Tencent Technology Co., LTD
China

Xifeng Gao*
xifgao@global.tencent.com
LightSpeed Studios
USA

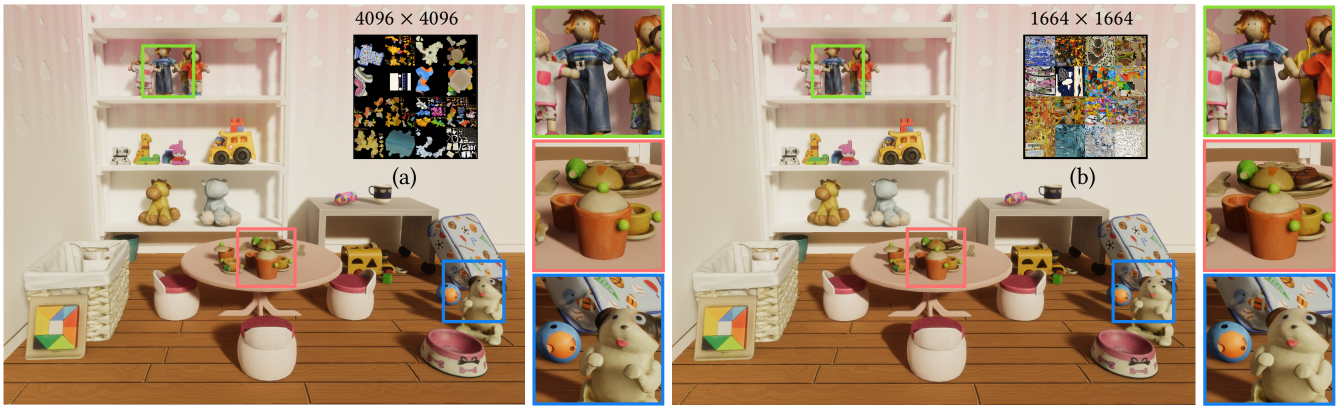

Figure 1: A corner of a bedroom full of complex 3D models with the original texture shown in (a). We use our method to compress the texture of each model and then arrange them together (b). We use three zoom-in views to highlight the marginal visual modification due to our method. Compared with the input, we achieve a texture compression ratio of 83.50% while preserving the visual appearance with the score of 43.19/0.987 as measured by PSNR/MS-SSIM.

ABSTRACT

Optimizing the memory footprint of 3D models can have a major impact on the user experiences during real-time rendering and streaming visualization, where the major memory overhead lies in the high-resolution texture data. In this work, we propose a robust and automatic pipeline to content-aware, lossy compression for texture atlas. The design of our solution lies in two observations: 1) mapping multiple surface patches to the same texture region is seamlessly compatible with the standard rendering pipeline, requiring no decompression before any usage; 2) a texture image has background regions and salient structural features, which can be handled separately to achieve a high compression rate. Accordingly, our method contains joint operations of image segmentation, re-meshing, UV unwrapping, and texture baking. To evaluate the efficacy of our approach, we batch-processed a dataset containing 100 models collected online. On average, our method achieves a texture atlas compression ratio of 81.41% with an averaged PSNR and MS-SSIM scores of 40.90 and 0.98, a marginal error in visual appearance.

CCS CONCEPTS

• **Computing methodologies → Texturing**; **Image compression**; **Mesh models**; **Rasterization**.

*Corresponding authors.

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0315-7/23/12…$15.00
https://doi.org/10.1145/3610548.3618150

KEYWORDS

Image Segmentation, Texture Compression, UV Re-Packing, Differentiable Rendering

ACM Reference Format:

Yuzhe Luo, Xiaogang Jin, Zherong Pan, Kui Wu, Qilong Kou, Xiajun Yang, and Xifeng Gao. 2023. Texture Atlas Compression Based on Repeated Content Removal. In *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3610548.3618150

1 INTRODUCTION

Textured meshes are the mainstream 3D model representation employed in the current GPU rendering pipeline. For efficient real-time rendering, a standard solution for artists is to map high-resolution textures onto the 3D mesh surfaces to encode rich appearance details. As a result, accessing these texture data can not only pose a major performance bottleneck for network data streaming, rendering algorithm, GPU bandwidth, and memory footprint, but also cause a large file size in disk or package size when releasing the developed applications.

Traditionally, 3D game assets are manually crafted by artists. During the modeling process, they can flexibly reuse mesh parts with the same texture data to save the final texture size for the completed model. On a parallel front, 3D reconstruction and AI-generated 3D models have recently surged as promising alternative approaches and garnered significant research attention. These methods use 3D scanning [Zhou and Koltun 2014], photogrammetry software [Agisoft 2023; Reality 2023], and inverse rendering [Hasselgren et al. 2021; Mildenhall et al. 2020] to digitalize real-world 3D assets, from which a mesh representation can either be directly generated or converted from iso-surfacing such as marching cube [Lorensen and Cline 1987] and the surface appearance can be recovered via visual-guided texture baking [Hasselgren et al. 2021]. However, the 3D meshes generated by these algorithms usually have high-resolution textures that contain repeated contents mapped to separate regions on the 3D surface, leading to excessively large texture data. To reduce the amount of texture data, one possible way for artists is to fine-tune these models by manually identifying the repeated texture contents and manipulating the texture, the UV mesh, and the 3D surface mesh. However, relying on manual processes is intractable not only because of the tedious and iterative editing, but also due to the rapidly growing amount of reconstructed or learned 3D models. Therefore, to compress the textures of algorithmically generated 3D models, designing an automatic and robust approach is urgent and highly demanded.

Given a 3D mesh with UV and texture, we propose a fully automatic pipeline that can greatly reduce its texture size by compressing its repeated contents, while incurring minimal changes to the visual appearance. Our key observation is that, by modifying the UV atlas, multiple surface patches can be mapped to the same texture region, while being seamlessly compatible with existing rendering pipeline without requiring any decoding algorithms or additional information. We further observe that the texture image can be considered as having two types of contents, i.e. salient foreground regions with rich information and background areas with typically monotonic color, which can be processed separately to achieve a high compression rate. Accordingly, our algorithm has three major phases: *repeated salient region removal*, *background compression*, and *new texture generation*.

Our approach can automatically compress the texture of any 3D model, with a single user-controlled parameter ϵ to balance the compression ratio and the visual quality. We batch-tested the proposed method on 110 models acquired through 3D scanning, where we achieved a texture compression ratio of 81.41% and an averaged PSNR and MS-SSIM of 40.90 and 0.98, respectively, compared to the input models. A complex scene processed using our method is illustrated in Figure 1. We attach as supplementary the tested dataset and the results, including meshes, textures, statistics, and video clips of the final rendered results of each model in the dataset. The executable program can be found here[1].

To sum up, the contributions of our work include:

- We introduce the problem formulation of content-aware, standard renderer-compatible texture compression for general 3D models via removing repeated texture contents.
- To the best of our knowledge, we are the first to propose an automatic and practical technical pipeline to reduce the texture size by designing dedicated strategies for the foreground and background texture contents, respectively, while synchronizing the changes among the texture, the UV map and the 3D mesh.

2 RELATED WORK

Our method compresses the texture of a 3D model by removing repeated texture contents, which is highly related to topics in image and texture compression, feature extraction, and image matching, as briefly reviewed below.

2.1 Image and Texture Compression

Digital image compression is a highly active field of research. Many traditional methods such as JPEG [Wallace 1991] utilize spectral redundancy between neighboring samples within an image to design both the compressor and the decompressor, as summarized in [Hussain et al. 2018]. More recently, innovative and enhanced image compression formats, such as JPEG XL [Alakuijala et al. 2019] have been proposed to mitigate the potential artifacts of JPEG compression. Meanwhile, a large number of neural image compression techniques have emerged, capitalizing on advancements in neural networks for image perception quality. These techniques employ CNN [Zhao et al. 2019], RNN [Toderici et al. 2017], and GAN [Rippel and Bourdev 2017; Tschannen et al. 2018] to recover the high-level repetitive contents. However, these methods compress through global encoders and decoders and do not support random access, so they are not suitable for real-time rendering on GPUs.

Based on the demand for efficient random-access texture compression methods for real-time applications on GPUs, a series of algorithms that exploit data locality for block coding have been proposed, including BC1-BC7 in DirectX [Microsoft 2020], Ericsson texture compression (ETC1/ETC2) [Ström and Akenine-Möller 2005; Ström and Pettersson 2007] in OpenGL ES, and ASTC [Nystad et al. 2012]. QuickETC2 [Nah 2020] utilizes the luma difference of the image blocks for the preprocessing of compression mode selection to

[1] https://texture-atlas-compression.github.io

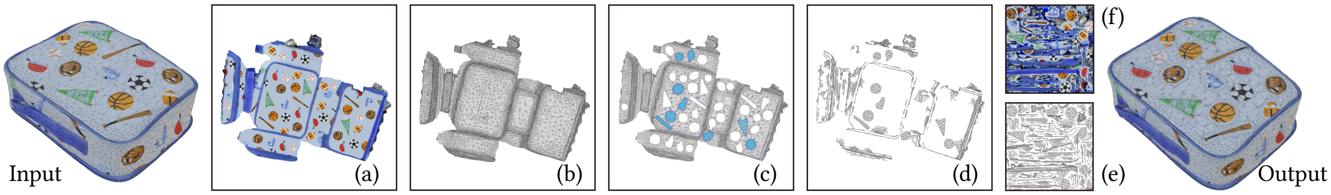

Figure 2: Pipeline overview: The input to our method is a 3D model with a texture (a) and a UV map (b). We first detect and remove the repetitive salient feature regions by updating the UV map with the representative region colored in blue (c). We then compress the background texture to generate a new UV map by removing triangles with repetitive contents (d). Finally, we create a new UV atlas by cutting and re-packing (e). We further bake a new texture via differentiable rendering (f).

reduce the unnecessary compression overhead, thus achieving accelerated ETC compression. GST [Krajcevski et al. 2016] introduces a novel supercompression method based on the state-of-the-art entropy encoding technique known as ANS for already compressed textures, designed for endpoint compressed formats like DXT and PVRTC. Vaidyanathan et al. [2023] introduce a neural compression technique specifically designed for material textures by integrating techniques from GPU texture compression as well as neural image compression. They compress the texture and the mipmap chain together and use a small neural network to decompress them. Wang et al. [2008] proposed a search-based block encoding technique, where large image blocks are affine-transformed to match repeated blocks, which again requires additional changes to the standard GPU rendering pipeline. Peyré [2009] presents a generative model for textures that uses a local sparse description of the image content. Li and Wand [2015] introduce approximate translational building blocks for unsupervised image decomposition and synthesis. Nvidia introduced the nvCOMP library [NVIDIA 2022] to achieve fast lossless compression of data on the GPU. However, these prior works do not consider the textures mapped onto 3D surfaces, where multiple factors, including texture, UV mesh, and 3D mesh, will be jointly affected during the texture compression.

Carr and Hart [2002] developed seamless meshed atlases for solid texturing. Purnomo et al. [2004] introduced a parametric approach to building seamless atlases based on quadrilateral charts. Box Cutter [Limper et al. 2018] introduced a method to improve atlas packing efficiency without changing UV distortion. But they do not take into account the semantic information carried by the texture itself for compression. Wei et al. [2008] proposed an approach to compress procedural-like material texture, but requires an additional control map and does not consider the relationship between 3D mesh surface and the texture. We propose the first-ever pipeline to reduce texture size by removing semantic-level repetitions from textures while jointly modifying the 3D mesh and UV mapping. We note that our approach is not suitable for processing textures generated by texture synthesis techniques.

2.2 Feature Extraction and Matching

The first phase in our pipeline involves detecting texture patches containing salient information. Salient object detection and segmentation is a well-established research area in computer vision. However, it was not until recently that fine-grained object boundaries could be identified and segmented using a series of novel deep network architectures, including boundary-aware attention feedback network [Feng et al. 2019], pixel-wise contextual attention network [Liu et al. 2018], and GAN [Mukherjee et al. 2019]. Recently, the Segment Anything Model (SAM) [Kirillov et al. 2023] has made a breakthrough in terms of both robustness and quality. It is trained on a large-scale segmentation dataset and can be adapted to the texture domain, as demonstrated in our work.

Switching gears to feature matching, the Kanade-Lucas-Tomasi (KLT) algorithm [Lucas and Kanade 1981; Shi et al. 1994] was proposed to register two rectangular image blocks with similar contents, by locally optimizing the relative affine transformation and maximizing a similarity score. Leung and Malik [1996] proposed a comprehensive algorithmic pipeline for detecting, localizing, and grouping instances of repeated scene elements. Their algorithm uses a graph with individual image elements as nodes and their affine transforms as edges. However, these algorithms focus on rectangular image blocks. Berg et al. [2005] proposed to match only correspondence points instead of an entire image block. Cheng et al. [2010] proposed a boundary band matching method that requires user interaction to detect duplicate objects. Recent deep-learning-based techniques [Aberman et al. 2018] can detect corresponding points without exhaustive search. By incorporating SAM and KLT algorithms into our pipeline, we derive robust performance in redundant feature removal. Although KLT is limited to affine transformations, we found it sufficient to match the texture patches in our dataset. Note that other feature matching algorithms [Berg et al. 2005; Cheng et al. 2010; Kong et al. 2013] can be used within our pipeline if non-rigid matching is desired.

3 METHOD

The input to our method is a general 3D model with provided UV atlas and texture image. Our goal is to output a new model with updated mesh, UV, and texture, such that the new texture size is as small as possible, while the output model is visually similar to the input from all viewing directions.

As illustrated in Figure 2, our approach has three major phases. We first detect and remove the repetitive salient feature regions by mapping the corresponding 3D surface patches to the same UV region, resulting in concordant updates of mesh, UV, and texture (Figure 2(c), Section 3.1). In our second phase, we compress the background texture by processing the triangles that are not in the salient regions. Specifically, we cluster the non-salient triangles with small color variations into groups, identify a representative triangle for each group, and remap all 3D triangles within the same group to the UV region of one representative triangle (Figure 2(d), Section 3.2). At this point, the remapped texture image usually contains many void spaces that can be squeezed out. Our last phase creates a new UV atlas by cutting and re-packing (Figure 2(e), Section 3.3). We then bake a new texture image by adapting a differentiable rendering technique to remove seam artifacts (Figure 2(f), Section 3.3). The pseudocode for the first two phases are attached as supplementary material.

3.1 Repetitive Salient Feature Removal

In this work, we refer to a salient feature as a semantically meaningful structural pattern that can span across multiple triangles, as shown in Figure 2(ab). To remove such redundant features, multiple challenges arise: 1) Unlike 2D images, textures for a 3D model are *not* continuous at UV patch boundaries; 2) robustly detecting the feature contour for general 2D images is a classical challenging problem in computer vision; and 3) the texture content is mapped back to the 3D surface, so changes to the texture content should be synchronized by corresponding changes to UV atlas and 3D meshes for visual consistency. To detect and remove the redundant salient features while addressing the aforementioned challenges, we introduce an automatic strategy consisting of the following four steps.

Step 1: Texture Dilation. To handle the texture content discontinuity across UV chart boundaries, we dilate the texture by adopting the traveler's map method [González and Patow 2009]. As illustrated in Figure 3, we first find all the UV boundary edges corresponding to cuts, i.e. the corresponding 3D mesh edges are not on the boundary. We then dilate the UV patches by extruding every UV vertex v on such boundaries along its normal direction n by a distance d_D. For each boundary edge $e(v_i, v_j)$, we can construct a quadrilateral region formed by four vertices, denoted as $M(v_i, v_j, v_j + n_j d_D, v_i + n_i d_D)$. We further dilate the texture image in the quadrilateral region by copying the texture data from the other UV patch. Assuming the corresponding edge on the other patch is e', e and e' can be aligned via an affine transformation $T_{e \to e'}$. We can thus recover a continuous local copy of cross-chart contents by copying the texture data from $T_{e \to e'} x$ back to $x \in M(v_i, v_j, v_j + n_j d_D, v_i + n_i d_D)$. An example of dilation is shown in Figure 9.

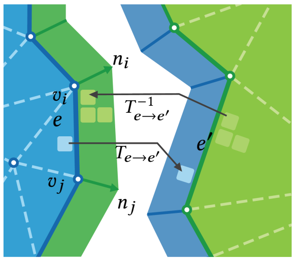

Figure 3: The void space of the texture adjacent to the edge e (green) is filled up by copying from the texture content of another UV patch neighboring the corresponding edge e'.

Step 2: Feature Extraction. We adopt the recently proposed SAM model [Kirillov et al. 2023] that has an excellent zero-shot transferability to novel domains, which outputs a set of texture regions denoted as $\mathcal{M}^f$, representing the set of foreground features of the dilated texture M. Each region $M_i \in \mathcal{M}^f$ is equipped with a segmentation mask marking all the pixels belonging to the salient feature. Note that, for any $M_i \in \mathcal{M}^f$, it could overlap with another region $M_j \in \mathcal{M}^f$ or even fully contains it, which means a pixel can belong to multiple regions. As shown in Figure 4, $\mathcal{M}^f$ may also contain non-salient features and incorrect segmentation. Based on the assumption that a salient region should have an appropriate size and a large color variation, we consider a region $M_i \in \mathcal{M}^f$ as invalid and discard it from $\mathcal{M}^f$ if it does not satisfy the two conditions below:

- $|M_i|/|M| \in [\underline{\epsilon}, \bar{\epsilon}]$, where $|\bullet|$ is the number of pixels in $\bullet$ and M without subscripts denotes the entire texture image;
- $\sigma(M_i) \geq \epsilon_\sigma$, where $\sigma(M_i)$ is the standard hue variation of pixel colors within M_i.

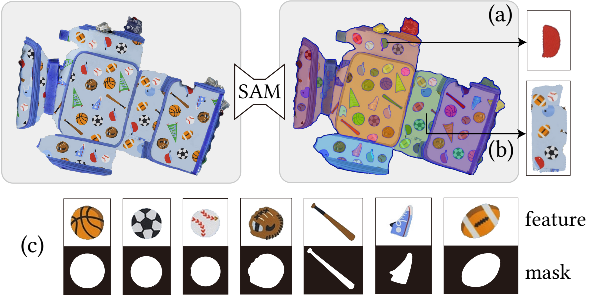

Figure 4: The dilated texture is first preprocessed by applying the SAM model. Further filtering is then applied to each candidate region M_i. A region is invalid if it has small color variations (a) or inappropriate size (b); otherwise, it is valid (c). For each valid region, we show the salient feature on the top row and the corresponding mask at the bottom.

Step 3: Repetitive Feature Detection. Given $\mathcal{M}^f$, we move on to match all regions with repetitive contents. We iterate through each pair $M_i, M_j \in \mathcal{M}^f$ and attempt to register M_i to M_j via the KLT approach introduced in [Lucas and Kanade 1981; Shi et al. 1994; Wang et al. 2008]. Instead of matching on the full rectangular region, as done in the original algorithm, our computational domain is the salient irregular region M_i. For completeness, we briefly describe the approach below.

Given M_i and M_j, we compute an affine transformation T_{ij}, such that for any texel $x \in M_i$, the color difference between $M[x]$ and $M[T_{ij}x]$ is small. To this end, the following image-matching objective $O(M_i, T_{ij})$ is minimized:

$$\mathrm{argmin}_{T_{ij}} O(M_i, T_{ij}) \triangleq \frac{\sum_{x \in M_i} \|M[x] - M[T_{ij}x]\|^2}{\sigma(M_i)^\alpha + \lambda}, \tag{1}$$

where coefficients α and λ are perceptual factors to better preserve low-contrast features in relatively smooth regions [Wang et al. 2008]. Note that M_j does not participate in the optimization process, but it is used to offer an initialization of T_{ij} through the registration of the bounding boxes of M_i and M_j. Equation 1 is solved using Newton's method (refer to [Wang et al. 2008] for details). We consider M_i successfully registered to M_j whenever $O(M_i, T_{ij}) \leq \epsilon_o |M_i|$ on convergence.

Step 4: Local Re-meshing. With all the salient regions matched, the region set $\mathcal{M}^f$ is written as the disjoint union: $\mathcal{M}^f = \bigcup_i \mathcal{M}^f_i$, where $\mathcal{M}^f_i$ is the region subset with the same repetitive contents that are successfully matched. We select the representative region

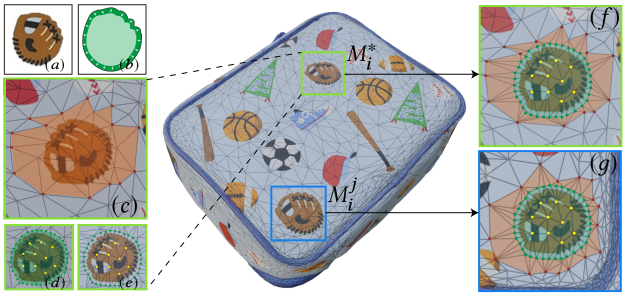

Figure 5: Given the representative region M_i^* (a), we first dilate M_i^* to get an enlarged mask (b). A local UV mesh (orange) $\mathcal{T}_E(M_i^*)$ is selected, covering M_i^* (c), and a set of boundary (green) and interior (yellow) samples are then computed on M_i^* (d). This is followed by creating a new mesh $\mathcal{T}_C(M_i^*)$ using CDT with both boundary and interior samples as hard constraints (e) and another CDT with both the boundary samples and the boundary loop of $\mathcal{T}_E(M_i^*)$ as hard constraints (f). To unify UV coordinates for other feature regions, e.g. M_i^j, we will replace the local UV patch of M_i^j with $T_{ij}\mathcal{T}_C(M_i^*)$ (green), and re-mesh the gap region $\mathcal{T}_E(M_i^j) - T_{ij}\mathcal{T}_C(M_i^*)$ (orange) (g).

$M_i^* \in \mathcal{M}_i^f$ that has the smallest registration error with other regions. and refer all other UV patches in $\mathcal{M}_i^f$ to M_i^* for texture compression. To this end, a widely used technique is re-indexing, where an image codebook is maintained and all relevant patches are replaced with an index into the codebook [Hussain et al. 2018; Wang et al. 2008]. However, this strategy requires an additional codebook indexing procedure, which can considerably complicate the existing rendering pipelines. Instead, we propose a re-meshing algorithm that unifies the UV mapping of all the duplication patches by slightly modifying the local connectivity around the patches. Our approach generates 3D models with data representation fully compatible with the standard rendering pipeline.

Taking $M_i^j \in \mathcal{M}_i^f$ for example, which is to be referred to by $M_i^* \in \mathcal{M}_i^f$ (Figure 5(a)), we first dilate M_i^* by d_M pixels to ensure that the salient content is fully contained in the mask (Figure 5(b)), which also ensures that the transformed mask $T_{ij}M_i^*$ fully contains M_i^j. Due to the possible inaccuracy near the feature boundaries of the SAM segmentation, this step is used to avoid the cases where there is missing coverage of salient features during re-meshing. Next, as highlighted in the orange region in Figure 5(c), we identify a local UV mesh enclosing M_i^*, denoted as $\mathcal{T}_E(M_i^*)$. We compute an averaged edge length of $\mathcal{T}_E(M_i^*)$, denoted as $|\bar{e}|$, and sample the boundary of M_i^* at a regular interval of $0.5|\bar{e}|$. We further sample the interior of M_i^* using Poisson disk sampling with radius $|\bar{e}|$ (Figure 5(d)). Given the set of boundary and interior samples, we adopt the Constrained Delaunay Triangulation (CDT) [Žalik and Kolingerová 2003] to create a mesh $\mathcal{T}_C(M_i^*)$ for M_i^* (Figure 5(e)). We further perform a CDT to fill the following gap (Figure 5(f)):

$$\mathcal{T}_E(M_i^*) - \mathcal{T}_C(M_i^*). \tag{2}$$

In order to refer M_i^j to M_i^* in the UV space, we transform $\mathcal{T}_C(M_i^*)$ by T_{ij} computed from our previous step, denoted as $T_{ij}\mathcal{T}_C(M_i^*)$. Next, we identify a local UV mesh enclosing the transformed salient region, denoted as $\mathcal{T}_E(M_i^j)$. We remove all the triangles in $\mathcal{T}_E(M_i^j)$, insert the transformed local patch $T_{ij}\mathcal{T}_C(M_i^*)$, and run another CDT to fill the following gap:

$$\mathcal{T}_E(M_i^j) - T_{ij}\mathcal{T}_C(M_i^*), \tag{3}$$

which is illustrated as the triangles in orange in Figure 5(g). Note that, all the newly created meshes in the above steps will be reflected on the 3D mesh as well, where the 3D vertex coordinates of newly introduced vertices are copied from the input mesh through barycentric interpolation in UV space.

3.2 Background Compression

After the repeated structural contents are removed, we proceed to our second stage, compressing the background by removing triangles containing repeated background colors. We denote the background of the texture as $\mathcal{M}^b = M - \cup_i(\cup_j T_{ij}\mathcal{T}_C(M_i^*) \cup \mathcal{T}_C(M_i^*))$, representing the texture regions excluding the compressed salient features. Unlike foreground features that can span multiple triangles, the background is often composed of constant-color regions. Therefore, we consider background at the triangle element level. Specifically, we perform the following two steps: *clustering* all triangles in groups with (nearly) constant color; *compressing* all the triangles in a group by referring their UV coordinates to one triangle.

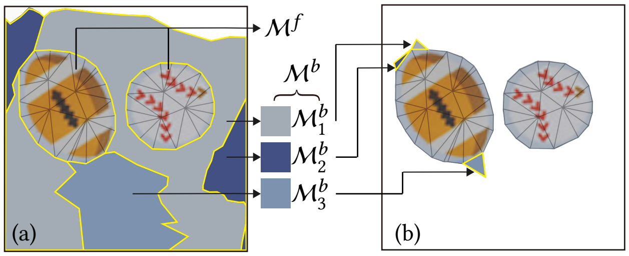

Figure 6: We separate the texture domain (a) into foreground salient features $\mathcal{M}^f$ and background triangles $\mathcal{M}^b$. We further cluster background triangles into constant-color groups $\mathcal{M}^b_{1,2,3}$. We pick one triangle M_j^* to represent $\mathcal{M}_j^b$ leading to the shortest cut length (b). Oftentimes, our greedy algorithm will find the triangle that is connected to some foreground salient regions.

Step 1: Clustering. Each triangle in the background occupies a texture region consisting of a set of texels, and we compute the mean and variance of their colors in HSV color space for closer alignment with human perception of color attributes. Triangles with excessively large color variance should be left uncompressed, while other triangles are considered to have a nearly constant color if they satisfy the following condition:

$$\sigma(H) < 100\epsilon \ \wedge\ \sigma(S) < 0.5\epsilon \ \wedge\ \sigma(V) < 3\epsilon, \tag{4}$$

where $\sigma(\bullet)$ is the color variance of the color channel $\bullet$, $H \in [0, 360)$, $S \in [0, 1]$, $V \in [0, 1]$, and ϵ is the only user-controllable parameter. All the triangles with nearly constant color will typically form multiple disconnected regions. If a region is tiny, then compressing it will not increase the compression ratio much but create cuts, which might not be beneficial for rendering. Therefore, we choose to filter out those tiny regions as well if their covered texture area is smaller than a parameter s. For all the rest of the triangles with near constant color, we perform a mean-shift clustering [Comaniciu and Meer 2002] in the HSV feature space with the bandwidth threshold set to 0.5ϵ. Since ϵ controls both the triangles participating in clustering and the clustering bandwidth, essentially, it balances the number of groups that affects the final texture compression ratio against the visual similarity. After clustering, the constant-color subset of $\mathcal{M}^b$ endows a disjoint union: $\bigcup_i \mathcal{M}_i^b \subseteq \mathcal{M}^b$.

Step 2: Compression. For each cluster of triangles $\mathcal{M}_i^b$, we consider them as all having the same color. Similar to compressing the foreground, we refer all the triangles to one representative $M_i^* \in \mathcal{M}_i^b$. However, unlike foreground regions that span many triangles, the background triangles can be randomly disseminated across the texture space, and treating each triangle as a separate UV patch can create many cuts, introducing additional seam artifacts and inadvertently affecting the memory footprint. To reduce the unnecessary cuts, we observe that if triangles in the same cluster $\mathcal{M}_i^b$ are already connected to a non-background patch, then they do not need to be cut apart. Further, if a background triangle is connected to a foreground region, then such connecting edges do not need to be cut either. Therefore, we adopt a greedy algorithm to search for the representative triangle as the one in $\mathcal{M}_i^b$ with the largest area while satisfying the above two rules, as illustrated in Figure 6.

3.3 New Texture Generation

After removing the repetitive texture content, the UV mesh may contain large void spaces. During this final stage, we eliminate these voids by generating a new UV atlas, and then bake a new texture image to replicate the visual appearance of the input as closely as possible.

Step 1: UV Packing. We adopt the standard UV atlas generation pipeline in *XAtlas* [Young 2023], which involves cutting, parameterization, and packing. Since our goal is to replicate the visual appearance as closely as possible, we skip the parameterization substep of *XAtlas* by default to avoid distortions to the input UV map, and compute only rigid transformations during the UV packing.

Step 2: Texture Baking. After regenerating UV atlas, we can fill in the texture image by interpolating the input texture. However, since we have cut the mesh along multiple boundaries for compression, a naive interpolation scheme can lead to discontinuous colors along new UV patch boundaries (Figure 11(a)). We propose to eliminate this problem by a global texture baking using the differentiable-rendering-based optimization framework [Hasselgren et al. 2021]. Starting from the interpolated texture as an initialization, we perform 5,000 iterations of stochastic gradient descent to optimize the compressed texture image. During each iteration, we randomly generate a camera orientation, render both the uncompressed and compressed models, and then compute the l_1-pixel-wise distance as our objective function. We further found that the camera poses sampled on a global spherical surface as in [Hasselgren et al. 2021] prevent texture optimization in geometrically self-occluded regions (Figure 11(b)). Instead, we sample the camera position at a distance d_N along the normal direction of a randomly picked triangle to capture as many texture details as possible (Figure 11(c)). This strategy is simple yet effective, considerably improving the visual appearance metrics: PSNR and MS-SSIM.

3.4 Implementation Details

We implement the proposed pipeline in Python with the smallest SAM model provided by the authors [Kirillov et al. 2023] (*vit_b*) for salient segmentation. We did not use a larger SAM model due to the limitation of our computational resources. We employ CGAL [Brönnimann et al. 2017] for CDT in the local re-meshing step, *XAtlas* [Young 2023] for UV cutting and packing, and the codebase of NVdiffmodeling [Hasselgren et al. 2021] for texture baking.

Hyper Parameters. We use the following default parameter settings: $d_D = 0.02\Delta$, where Δ is the width of the input texture and d_D controls the dilation distance for the traveler's map; $\underline{\epsilon} = 0.001$ and $\bar{\epsilon} = 0.25$ filter out extremely small and large salient patches for repetitive feature detection; $\epsilon_\sigma = 2.0$ determines if a patch is salient or not; $\alpha = 1.5$, $\lambda = 1.0$, and $\epsilon_o = 0.01$ are chosen as suggested in [Wang et al. 2008] for robust salient feature registration; $d_M = 2$ for mask dilation distance that is used during local re-meshing.

s is used to filter out those small connected regions of a cluster during background compression to avoid unnecessary cuts, where a large s may affect the final texture compression rate as illustrated in Figure 10 left. d_N is the distance between cameras and the mesh surface during texture baking, balancing the captured texture details and the texture continuity near seams (Figure 10 right). We chose $s = 0.005|M|$ and $d_N = 2.0$ as default values.

As described in Section 1 and Section 3.2, ϵ is the only parameter we expose to users since it plays a critical role in determining the number of clusters for the background compression, which is also intuitive for users to tune. We highlight in Figure 12 the effect of choosing different values for ϵ. A higher value of ϵ leads to smaller texture images but larger visual discrepancy. By default, we set $\epsilon = 0.10$ for all the experiments.

4 EXPERIMENTS

All of the experiments are run on a desktop machine equipped with an RTX2060 GPU and an i7-9700 CPU, where both SAM and NVdiffmodeling are performed on the GPU and all the other steps of our approach are run with a single CPU thread. We provide a computational timing statistics of the major steps of our approach on the entire dataset in the supplemental document.

Dataset. We collect a dataset containing 110 3D reconstructed models, where 100 are manually downloaded from the Gazebo platform [Robotics 2023] and 10 from OmniObject3D [Wu et al. 2023]. We then downsample the texture of each model to 1024^2 due to the GPU resource limitation to run SAM. Table 1 on page 7 shows the statistics of the examples in the paper and Figure 14 provides the corresponding visual gallery.

Evaluation Metrics. We employ four metrics to evaluate the efficacy of our method from various aspects. The ratio between the number of texels of the input texture and that of the output texture, denoted as CR. We evaluate the appearance quality of 3D models using the generated texture via PSNR and MS-SSIM [Wang et al. 2003], which is computed by sampling a set of 50 camera views

Table 1: Statistics for models shown in Figures 2, 9, 7, 11 and 12, including numbers of vertices/faces for the input mesh $\#V/\#F$ and the output $\#V'/\#F'$, the texture compression rate CR, PSNR, MS-SSIM, Hausdorff Distance (HD denotes the distance with the unit of 10^{-3}) between the input and the output meshes, and the computational time T.

Models	$\#V/\#F$	$\#V'/\#F'$	CR	PSNR	MS-SSIM	HD	$T(s)$
Fig. 2	$7.1K/14.2K$	$8.8K/16.5K$	78.14%	39.82	0.97	4.47	529
Fig. 7 (a)	$15.6K/31.5K$	$15.6K/31.5K$	86.94%	41.61	0.96	0.00	577
Fig. 7 (b)	$3.8K/7.7K$	$3.8K/7.7K$	79.74%	41.81	0.98	0.00	418
Fig. 9	$5.4K/10.9K$	$5.6K/11.2K$	85.94%	38.24	0.97	0.94	503
Fig. 11	$19.0K/38.0K$	$19.0K/38.0K$	85.94%	39.07	0.96	0.00	692
Fig. 12	$7.8K/15.6K$	$7.8K/15.6K$	93.80%	38.44	0.96	0.00	668

from a sphere surface of radius 2.5 centered around the model that is resized into the box $[-1, 1]^3$. We used the physically based render with a single light source in NVdiffmodeling [Hasselgren et al. 2021] to create the rendering for each camera view. Since our salient feature compression can potentially change the 3D mesh, we further compute the Hausdorff distance using Metro [Cignoni et al. 1998]. We aim to generate results with small texture size and Hausdorff distance while maintaining high PSNR and MS-SSIM values.

Results. Over the tested dataset, our method can significantly reduce the texture size, achieving an ave./min./max. compression ratios of 81.41%/24.33%/99.06%. The visual discrepancy against the uncompressed ground-truth mesh is rather small, with the ave./min./max. PSNR and MS-SSIM at 40.90/35.35/54.16 and 0.98/0.91/0.99, respectively. Our processed models have small geometric differences from the inputs, with the ave./min./max. Hausdorff distance of $1.6 \times 10^{-4}/0.00/5.80 \times 10^{-3}$. Figure 15 shows a set of results of the processed dataset. We provide all the results and the full statistics in the supplementary materials.

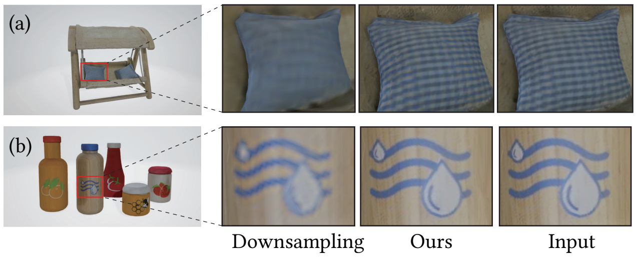

Figure 7: Our method compares favorably with direct texture downsampling. Under the same texture size, direct downsampling can lead to significant content blurring, while our method maintains all the visual details.

Comparison with Downsampling. We compare our method with direct texture downsampling, a straightforward alternative to texture compression. We match the compression ratio of downsampling to each of our compressed texture, and compare the PSNR and MS-SSIM for the two methods. The ave./min./max. PSNR and SSIM for the downsampled meshes are 39.27/34.28/46.78 and 0.96/0.90/0.99, respectively, which are all lower than ours. It is worth noting that the values of the PSNR and MS-SSIM can be greatly affected by the sphere radius, from which the cameras are sampled. When a larger sampling radius is employed, both PSNR and MS-SSIM will be higher, since the number of texels taken by the 3D model over the entire screen is small. Figure 13 compares our approach with downsampling in terms of both metrics by varying the radius size, where the average PSNR and MS-SSIM values for the entire tested dataset are shown. As illustrated in the plot, our method gains more advantages as the camera gets closer to the model. Visually, as shown in Figure 7, the downsampled result exhibits a significant blurry appearance, while our method preserves the visual details well.

Compatibility with BC7. To demonstrate the superiority of our approach when combined with typical compression formats used in typical GPU rendering pipelines, such as BC7, we performed the following experiments using the Unity engine by 1) applying BC7 compression and enabling mip-map directly on the original input texture, denoted as *Ref*, 2) rounding the size of our generated texture to power of 2, applying BC7 and enabling mip-map on the rounded texture, denoted as *Ours*^{1}, 3) applying RGB 24 bit encoding directly on our generated texture, denoted as *Ours*^{2} and 4) repacking the input UV maps using XAtlas, rebaking a new texture, and using BC7 to compress it, denoted as *Repack*. As shown in the Table 2, our approach is fully compatible with current texture compression methods for GPU rendering, by obviously reducing the texture storage overhead with the cost of limited rendering quality degradation.

Table 2: Comparison statistics of our method with alternative pipelines when using typical compression formats, as compared to *Ref*.

Method	texture CR	memory CR	PSNR	MS-SSIM
Ours^1	90.12%	**89.49%**	45.08	0.9838
Ours^2	**91.49%**	79.79%	44.46	0.9841
Repack	39.42%	53.78%	46.03	0.9914

Timing. Over the entire dataset, the ave./min./max. time required to compress a model is $551.4s/382.2s/746.3s$. The average time consumption of each step of our method is summarized Figure 8. Currently, the two steps, i.e., differentiable rendering of texture baking and the standard mean shift [Cheng 1995] during the background compression, dominant 92.5% of the total computation. As a future work, we plan to design a PSNR-based stopping criteria for differentiable rendering and employ the fast mean shift [Jang and Jiang 2021] to re-engineer our implementation for faster speed.

5 CONCLUSIONS

We present the first automatic algorithm to compress texture atlas for any 3D models based on the removal of repeated texture content. Our method outputs the bundle of texture, UV map, and 3D mesh that is fully compatible with modern rendering pipelines, with controllable visual similarity against the input groundtruth. On a dataset of 3D scanned objects, we highlight that our method achieves a high texture compression ratio, while introducing a low visual discrepancy. We anticipate our approach will pave the

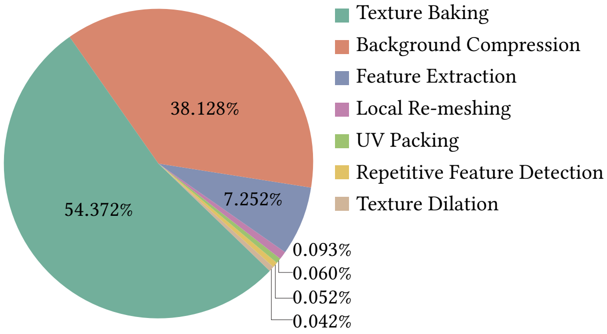

Figure 8: Averaged runtime statistics for the major steps of our pipeline over the entire dataset.

way for the practical applications of algorithmically generated 3D content.

Limitations. We do not address the issue when the distortion of UV map is large. This is because we found that the mapping distortion is typically small for 3D scanned models. However, for models with potentially large distortions, a pre-processing step could be used to unwrap UV in a low-distortion manner and then re-bake a new texture for the input. In addition, the employed traveler's map only partially resolves the discontinuity issue, since we cannot use an arbitrarily large chart dilation distance to prevent texture overlapping. To work around this issue, users can roughly mark the 3D surface region containing salient content, and then the marked region can be locally re-parameterized through existing approaches such as [Maggiordomo et al. 2023]. The parameterized region is then forwarded to SAM for segmentation. Lastly, our method removes triangles with repeated background colors without re-meshing, which may cause discontinuity in texture regions with gradual color changing. This could be potentially addressed through an optimization involving image gradients and we plan to address this issue as a future work.

ACKNOWLEDGMENTS

Xiaogang Jin was supported by the National Natural Science Foundation of China (Grant No. 62036010), and the Key Research and Development Program of Zhejiang Province (Grant No. 2023C01047). We would like to thank our colleagues from LightSpeed Studios, Fengquan Wang and Dong Li, for their project support.

REFERENCES

Kfir Aberman, Jing Liao, Mingyi Shi, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. 2018. Neural best-buddies: Sparse cross-domain correspondence. *ACM Transactions on Graphics* 37, 4 (2018), 1–14.

Agisoft. 2023. Agisoft: Agisoft Metashape: A 3D reconstruction software. Retrieved May, 2023 from https://www.agisoft.com/

Jyrki Alakuijala, Ruud Van Asseldonk, Sami Boukortt, Martin Bruse, Iulia-Maria Comşa, Moritz Firsching, Thomas Fischbacher, Evgenii Kliuchnikov, Sebastian Gomez, Robert Obryk, et al. 2019. JPEG XL next-generation image compression architecture and coding tools. In *Applications of Digital Image Processing XLII*, Vol. 11137. SPIE, 112–124.

Alexander C Berg, Tamara L Berg, and Jitendra Malik. 2005. Shape matching and object recognition using low distortion correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 1. IEEE, 26–33.

Hervé Brönnimann, Andreas Fabri, Geert-Jan Giezeman, Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Stefan Schirra. 2017. 2D and 3D Linear Geometry Kernel. https://doc.cgal.org/latest/Kernel_23/classKernel.html

Nathan A Carr and John C Hart. 2002. Meshed atlases for real-time procedural solid texturing. *ACM Transactions on Graphics* 21, 2 (2002), 106–131.

Ming-Ming Cheng, Fang-Lue Zhang, Niloy J Mitra, Xiaolei Huang, and Shi-Min Hu. 2010. Repfinder: finding approximately repeated scene elements for image editing. *ACM Transactions on Graphics* 29, 4 (2010), 1–8.

Yizong Cheng. 1995. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17, 8 (1995), 790–799. https://doi.org/10.1109/34.400568

P. Cignoni, C. Rocchini, and R. Scopigno. 1998. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum* 17, 2 (1998), 167–174.

Dorin Comaniciu and Peter Meer. 2002. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 5 (2002), 603–619.

Mengyang Feng, Huchuan Lu, and Errui Ding. 2019. Attentive feedback network for boundary-aware salient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1623–1632.

Francisco González and Gustavo Patow. 2009. Continuity mapping for multi-chart textures. In *ACM SIGGRAPH Asia 2009 papers*. 1–8.

Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. 2021. Appearance-driven automatic 3D model simplification. In *EGSR (DL)*. 85–97.

Abir Jaafar Hussain, Ali Al-Fayadh, and Naeem Radi. 2018. Image compression techniques: A survey in lossless and lossy algorithms. *Neurocomputing* 300 (2018), 44–69.

Jennifer Jang and Heinrich Jiang. 2021. MeanShift++: Extremely Fast Mode-Seeking With Applications to Segmentation and Object Tracking. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4100–4111. https://doi.org/10.1109/CVPR46437.2021.00409

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. 2023. Segment anything. *arXiv:2304.02643* (2023).

Yan Kong, Weiming Dong, Xing Mei, Xiaopeng Zhang, and Jean-Claude Paul. 2013. SimLocator: robust locator of similar objects in images. *The Visual Computer* 29 (2013), 861–870.

Pavel Krajcevski, Srihari Pratapa, and Dinesh Manocha. 2016. GST: GPU-decodable supercompressed textures. *ACM Transactions on Graphics* 35, 6 (2016), 1–10.

Thomas Leung and Jitendra Malik. 1996. Detecting, localizing and grouping repeated scene elements from an image. In *Computer Vision—ECCV'96: 4th European Conference on Computer Vision Cambridge, UK, April 15–18, 1996 Proceedings, Volume I 4*. Springer, 546–555.

Chuan Li and Michael Wand. 2015. Approximate translational building blocks for image decomposition and synthesis. *ACM Transactions on Graphics* 34, 5 (2015), 1–16.

Max Limper, Nicholas Vining, and Alla Sheffer. 2018. Box cutter: atlas refinement for efficient packing via void elimination. *ACM Trans. Graph.* 37, 4 (2018), 153–1.

Nian Liu, Junwei Han, and Ming-Hsuan Yang. 2018. Picanet: Learning pixel-wise contextual attention for saliency detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3089–3098.

William E. Lorensen and Harvey E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. Association for Computing Machinery, New York, NY, USA, 163–169. https://doi.org/10.1145/37401.37422

Bruce D Lucas and Takeo Kanade. 1981. An iterative image registration technique with an application to stereo vision. In *IJCAI'81: 7th international joint conference on Artificial intelligence*, Vol. 2. 674–679.

A Maggiordomo, P Cignoni, and M Tarini. 2023. Texture inpainting for photogrammetric models. *Computer Graphics Forum* 42 (2023).

Microsoft. 2020. Texture Block Compression in Direct3D 11. https://learn.microsoft.com/en-us/windows/win32/direct3d11/texture-block-compression-in-direct3d-11

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Prerana Mukherjee, Manoj Sharma, Megh Makwana, Ajay Pratap Singh, Avinash Upadhyay, Akkshita Trivedi, Brejesh Lall, and Santanu Chaudhury. 2019. DSAL-GAN: Denoising based saliency prediction with generative adversarial networks. *arXiv preprint arXiv:1904.01215* (2019).

Jae-Ho Nah. 2020. QuickETC2: Fast ETC2 texture compression using Luma differences. *ACM Transactions on Graphics* 39, 6 (2020), 1–10.

NVIDIA. 2022. NVCOMP. Retrieved Sep, 2023 from https://www.unrealengine.com/en-US/unreal-engine-5

Jörn Nystad, Anders Lassen, Andy Pomianowski, Sean Ellis, and Tom Olson. 2012. Adaptive scalable texture compression. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics Conference on High-Performance Graphics*. 105–114.

Gabriel Peyré. 2009. Sparse modeling of textures. *Journal of mathematical imaging and vision* 34 (2009), 17–31.

Budirijanto Purnomo, Jonathan D Cohen, and Subodh Kumar. 2004. Seamless texture atlases. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on*

Geometry processing. 65–74.

Capturing Reality. 2023. RealityCapture: A 3D reconstruction software. Retrieved May, 2023 from https://www.capturingreality.com/realitycapture

Oren Rippel and Lubomir Bourdev. 2017. Real-time adaptive image compression. In *International Conference on Machine Learning*. PMLR, 2922–2930.

Open Robotics. 2023. Gazebo. Retrieved May, 2023 from https://app.gazebosim.org/fuel/models

Jianbo Shi et al. 1994. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 593–600.

Jacob Ström and Tomas Akenine-Möller. 2005. i PACKMAN: High-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. 63–70.

Jacob Ström and Martin Pettersson. 2007. ETC 2: texture compression using invalid combinations. In *Graphics Hardware*, Vol. 7. 49–54.

George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. 2017. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 5306–5314.

Michael Tschannen, Eirikur Agustsson, and Mario Lucic. 2018. Deep generative models for distribution-preserving lossy compression. *Advances in Neural Information Processing Systems* 31 (2018), 5933—-5944.

Karthik Vaidyanathan, Marco Salvi, Bartlomiej Wronski, Tomas Akenine-Möller, Pontus Ebelin, and Aaron Lefohn. 2023. Random-Access Neural Compression of Material Textures. In *Proceedings of SIGGRAPH*.

Gregory K Wallace. 1991. The JPEG still picture compression standard. *Commun. ACM* 34, 4 (1991), 30–44.

Huamin Wang, Yonatan Wexler, Eyal Ofek, and Hugues Hoppe. 2008. Factoring repeated content within and among images. *ACM Transactions on Graphics* 27, 3 (2008), 1–10.

Z. Wang, E.P. Simoncelli, and A.C. Bovik. 2003. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Vol. 2. IEEE Computer Society, USA, 1398–1402 Vol.2.

Li-Yi Wei, Jianwei Han, Kun Zhou, Hujun Bao, Baining Guo, and Heung-Yeung Shum. 2008. Inverse texture synthesis. In *ACM SIGGRAPH 2008 papers*. 1–9.

Tong Wu, Jiarui Zhang, Xiao Fu, Yuxin Wang, Jiawei Ren, Liang Pan, Wayne Wu, Lei Yang, Jiaqi Wang, Chen Qian, et al. 2023. Omniobject3d: Large-vocabulary 3d object dataset for realistic perception, reconstruction and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 803–814.

Jonathan Young. 2023. Xatlas. Retrieved May, 2023 from https://github.com/jpcy/xatlas

Borut Žalik and Ivana Kolingerová. 2003. An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm. *International Journal of Geographical Information Science* 17, 2 (2003), 119–138.

Lijun Zhao, Huihui Bai, Anhong Wang, and Yao Zhao. 2019. Learning a virtual codec based on deep convolutional neural network to compress image. *Journal of Visual Communication and Image Representation* 63 (2019), 102589.

Qian-Yi Zhou and Vladlen Koltun. 2014. Color map optimization for 3d reconstruction with consumer depth cameras. *ACM Transactions on Graphics* 33, 4 (2014), 1–10.

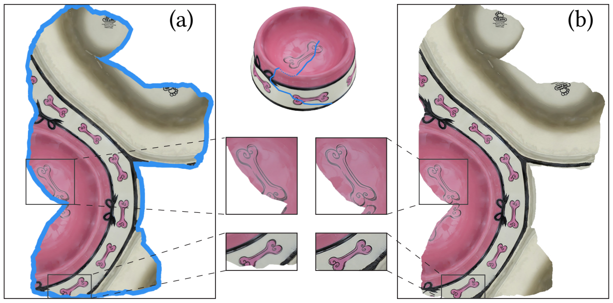

Figure 9: The dog bone patch on the bowl is cut apart (a) and our texture dilation technique extends the patch boundary to recover it (b).

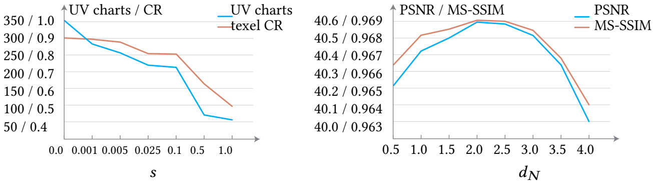

Figure 10: As s increases, the number of new UV charts after repacking decreases, so does the compression rate of the texture (CR). A large d_N leads to missing texture details for a particular mesh region while a small value may result in appearance discontinuity near texture seams.

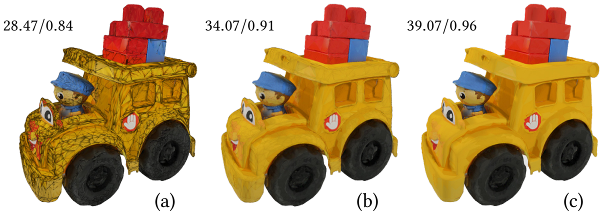

Figure 11: We compared the interpolation-based scheme (a) and the differentiable-rendering-based scheme (b). The texture quality is further improved by our normal-based camera pose sampler (c). The annotated numbers above each example are in the format of PSNR/MS-SSIM.

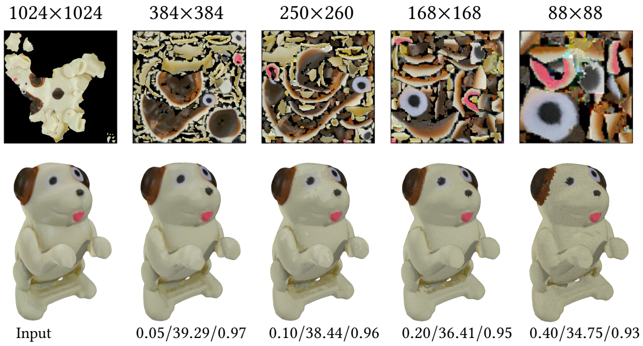

Figure 12: By choosing larger ϵ values, our method generates a smaller texture image but higher visual discrepancy (lower PSNR and MS-SSIM). The annotated numbers below each example are in the format of ϵ/PSNR/MS-SSIM.

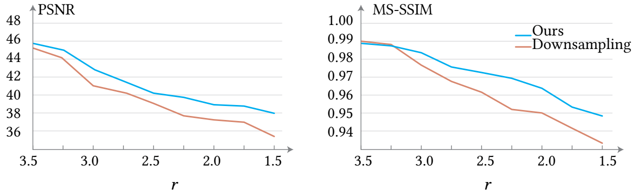

Figure 13: As the radius r of the sphere where the cameras are sampled from gradually decreases, which means a viewer is getting closer to the 3D model, our method has more obvious advantages than the downsampling method.

Figure 14: Visual gallery of all the models in the dataset.

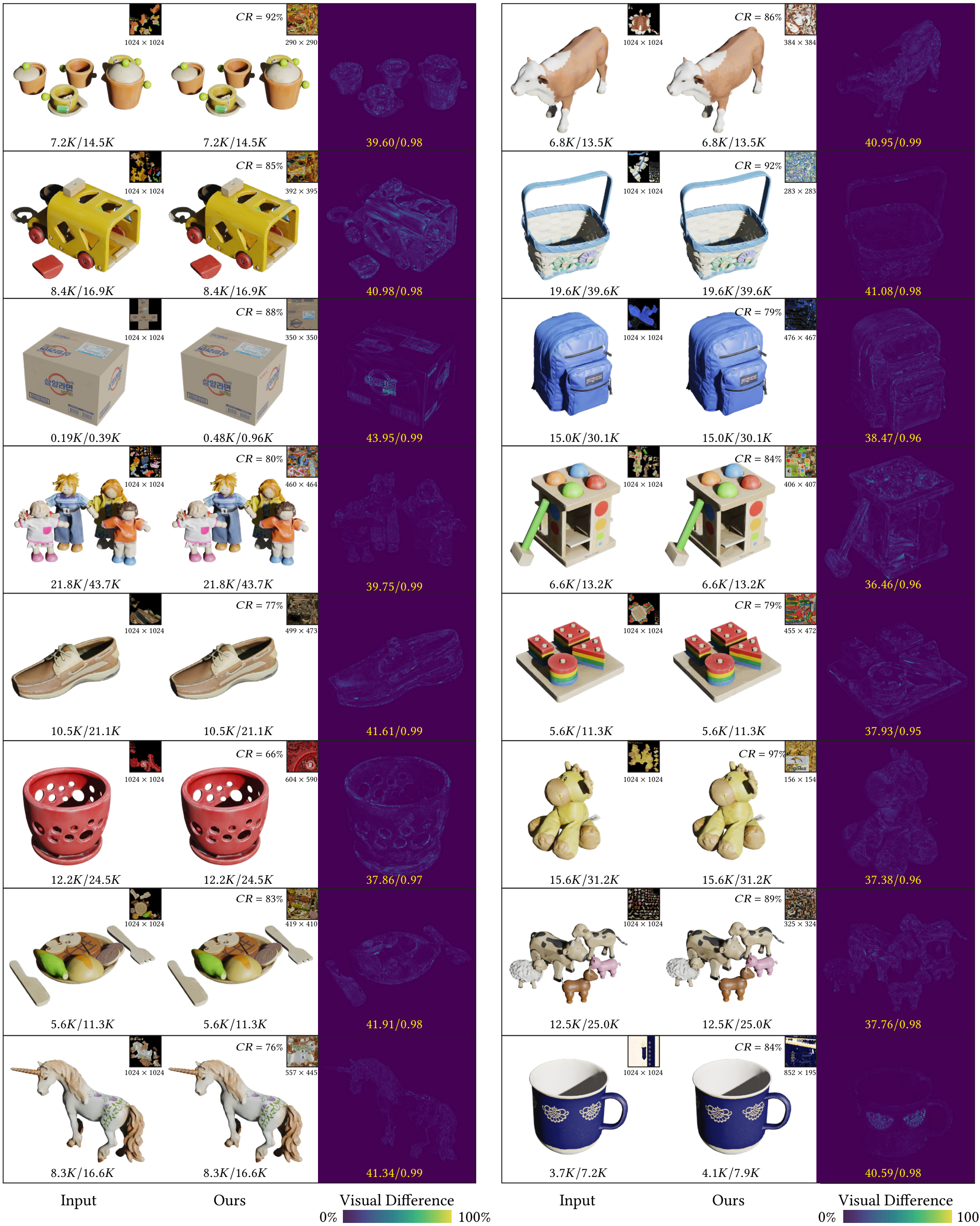

Figure 15: For each model, we provide information on the number of vertices and faces of the mesh, as well as the resolution of the texture, for both the input and the result. Additionally, we report the compression rate of the texture *CR* and the visual appearance preservation quality in terms of PSNR/MS-SSIM.