SUPPLEMENTARY TO RANDOM-ACCESS NEURAL COMPRESSION OF MATERIAL TEXTURES

B RENDERED QUALITY

In addition to the first figure in the main paper, Figures 14 and 15 show examples of rendered images using high-quality non-compressed textures, textures compressed with our method (using the lowest BPPC profile, NTC 0.2), and textures compressed with BC high. For the latter, the two highest-resolution mip levels were created through bilinear upscaling of the third mip level in order to obtain an iso-storage comparison to our method. The rendered images, supported by the error images and values, show that NTC achieves higher quality rendered images than the iso-storage version of BC high. This is especially noticeable in Figure 15, which also gives an indication that NTC does well on textures with text.

C ADDITIONAL QUANTITATIVE RESULTS

The following two subsections demonstrate how the quality of NTC's compressed images changes over the mipmap chain (Section C.1) and how it differs for various texture types (Section C.2).

C.1 Mipmap Quality

The performance of a compression technique can vary based on the frequency spectrum of the image and therefore perform differently across mip levels. In Figure 21, we compare the per-mip-level PSNR scores of NTC 0.2 to the BC high algorithm. Since there is a 16× difference in storage cost between the algorithms, an iso-storage comparison was conducted, resulting in the omission of the first two mip levels for BC compression. PSNR values with our method are either comparable to, or higher than BC depending on the mip level, except for mip levels two and three where NTC shows slightly worse PSNR scores.

C.2 Different Texture Type Compression Quality

Figure 20 presents the PSNR scores of NTC 0.20, computed on different types of textures in the material texture set, such as diffuse, normals, etc. Given the similarities in data content between certain texture types, such as ARM and ORM textures, as well as gloss and specular textures, the results for these pairs were concatenated. Additionally, texture types that occurred only once in our data set (see Section J) were excluded. The results indicate that our proposed method is able to compress different texture types at similar levels of quality.

D COMPRESSION ARTIFACTS

Every texture compression algorithm degrades quality differently. Particularly visible quality differences are called *compression artifacts* and we present a few typical examples in Figure 16. Block-based compression methods commonly exhibit visible block artifacts (inset a). Methods that rely on heavy quantization tend to exhibit banding artifacts, as illustrated in insets (b) and (c), and are often characterized by a visible discoloration towards green or purple hues, resulting from higher chroma quantization (inset b). These artifacts are highly perceptible to the human eye, and modern image compression techniques such as AVIF and JPEG XL, have prioritized their removal, by producing blurry images instead (inset d).

Since our feature vectors are quantized down to *two* or *four* bits per feature, we specifically check for the presence of banding artifacts by compressing a synthetic gradient texture and do not observe noticeable banding artifacts as shown in Figure 19. We attribute this to the combination of smooth, bilinear interpolation and the higher frequency learned interpolation (see Sections 4.1 and 4.3).

Figure 13 demonstrates the absence of visually objectionable artifacts with our compression method on an average case. In contrast, we observe the presence of block artifacts on the ARM texture with BC, especially for mip level 0. AVIF and JPEG XL produce sharper results than NTC for the diffuse texture at mip level 0 but are significantly blurrier or show some discoloration (JPEG XL) at mip level 3. This is likely because with these methods, different mip levels are compressed separately, and their spectral content does not necessarily reduce proportionally to the resolution. These observations make a strong case for jointly compressing mip levels as we do with NTC. Potentially, we could compress AVIF and JPEG XL by allocating different rates for each mip as well as each texture, while maintaining the overall storage constant. However, determining suitable rates can be challenging, particularly as it can be material specific. We do not include comparisons with heterogeneous rates for this reason, and also because our proposed method does not aim to compete directly with these image compression techniques.

On average, our method produces results that are a bit blurrier and sometimes less saturated than the uncompressed reference, but significantly better than BCx compression. There are, however, some more objectionable failure cases presented in Figure 12 in the main paper. In example a) we observe strong distortion of the normal map of the *Ticket Machine* texture. The compressed texture is very flat, mostly sparse, and our optimization procedure fails to reconstruct subtle details properly. In example b) (albedo of the *dragon atlas* material), we observe discoloration of the texture. Since our approach is specialized for each material, it can adapt to high frequency content, such as detailed normal maps, or large color variations. However, the most challenging materials have both kinds of features. In such cases, we cannot reconstruct both features equally well given the low BPPC rates. Typically, we observe that details and normal map features are favored by our optimization because of their higher variance, at the cost of other material textures. It is possible to balance the quality of material textures by adjusting the loss function (Section 4.5). In scenarios where the material textures have a fixed set of semantics, we can apply more robust texture specific optimization, such as a loss in chrominance space, or optimization based on appearance using differentiable renderering. The last two failure cases (c and d) in Figure 12 correspond to unusual data in the source materials. In example c) (*Pine Forest Ground* texture), the normal maps seem to be misaligned with the albedo maps, producing leakage of details between channels. Example d) (metalness map of the *Metal Plates* texture) shows strong banding in the source (reference) texture, present only in a single material channel, and our method blurs it and correlates with the other material channels.

Figures 17 and 18 show results of texture compression at other BPPC targets than the ones shown in the main paper. We note that the texture used to generate Figure 17 (*Pine Forest Ground*) is half the size (4096×4096) of the texture used to generate Figure 18 (*denim*) and the corresponding figure in the main paper. In Figure 17, we

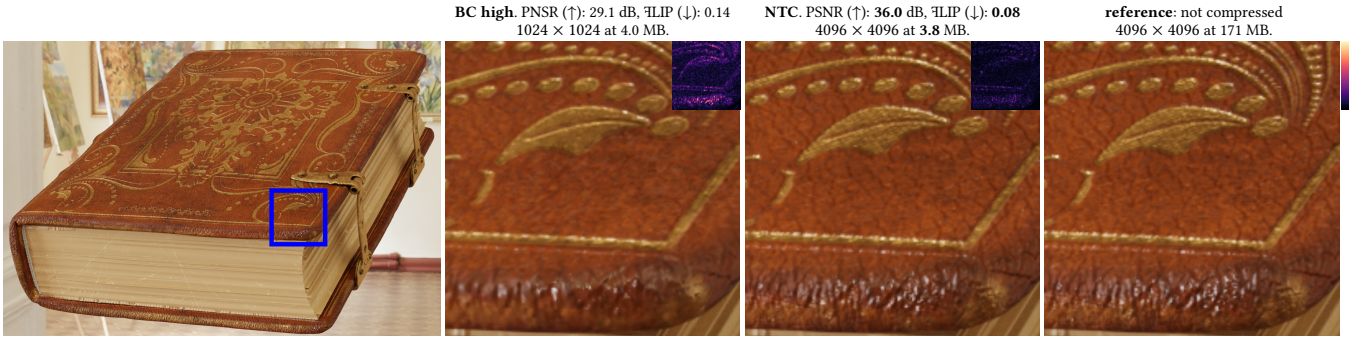

Fig. 14. A rendered image of a closed book. The cutouts demonstrate quality using, from left to right, GPU-based texture formats (BC high) at 1024 × 1024 resolution, our neural texture compression (NTC) , and high-quality reference textures. Note that NTC provides two additional mipmap levels over BC high, despite it using slightly less memory. The metrics, PSNR and ꟻLIP, were computed for the cutouts and are shown above the respective image. The ꟻLIP error images, whose brightness is proportional to error, are shown in the upper right corners.

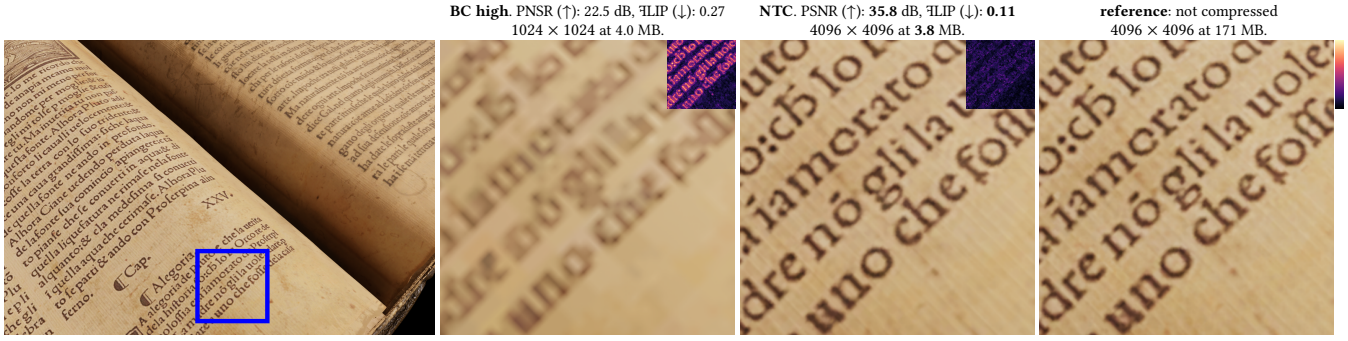

Fig. 15. A rendered image of an open book. See Figure 14's caption for details.

compare to the medium-low-rate JPEG XL and AVIF configurations, as well as the ASTC compressor (using tiles of size 12 × 12, as those lead to a mean BPPC closer to 0.50 for the evaluation data set compared to tiles of size 10 × 10). Figure 18 compares the medium-bitrate compressors. In the medium-bitrate case, it is difficult to spot any differences between the compressed textures and the reference. For the medium-low-rate case, differences become visible, most notably for the higher level mipmap where ASTC 12 × 12 and AVIF show block artifacts. Here, NTC 0.5 show slight color changes in the diffuse texture. An added challenge of this texture set is that the normal map is not aligned with the remaining textures.

E COMPARISON TO VECTOR QUANTIZATION

Vector quantization (VQ) is an alternate approach [78] to discretizing the features, where each cell in a feature grid maps to an entry in a learned codebook or dictionary. During inference, the feature vectors can be replaced by a codebook index stored per grid cell. Unfortunately, the codebook size grows exponentially with the bitrate, making it prohibitively expensive to learn a codebook for higher quality levels. Therefore, in order to compare VQ with scalar quantization (SQ), we limit the size of the dictionary to 256 entries and assume multiple dictionaries, such that the overall storage size is the same. We only apply vector quantization to the higher resolution grid G_0, which is quantized to a smaller number of bits, while G_1 always uses scalar quantization with 12 channels and 4 bits.

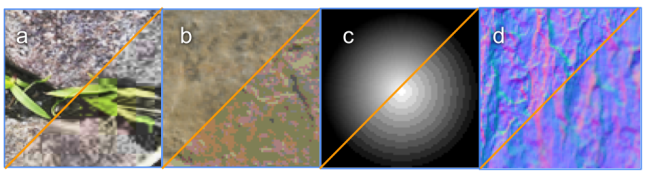

Fig. 16. Examples of typical compression artifacts. **a)** visible blocks **b)** posterization and discoloration **c)** gradient banding **d)** detail loss and bluriness.

Table 7 shows a comparison of SQ and VQ for a 4k texture using our lowest bitrate configuration (NTC 0.2) where each grid cell in G_0 stores 16 bits. In the case of SQ, we use 8 channels which are quantized to 2 bits while in the case of VQ, we use two 256-entries dictionaries, which are referenced by two 8-bit indices respectively. We compare SQ against two variants of VQ: VQ-8 has 8 channels per dictionary entry, which is similar to the size of the feature vector used in SQ, while VQ-16 uses 16 channels per dictionary entry and a correspondingly larger input layer in the decoder network. The PSNR for all three quantization options are within 0.5 dB of each other. VQ-8 has slightly lower PSNR than SQ, while VQ-16 has a slightly higher PSNR, but with a higher cost for the input layer of the network. The training time for both VQ-8 and VQ-16 is more than 2.5× that of SQ. Given its simplicity and the significantly shorter training time, we choose scalar quantization for compression.

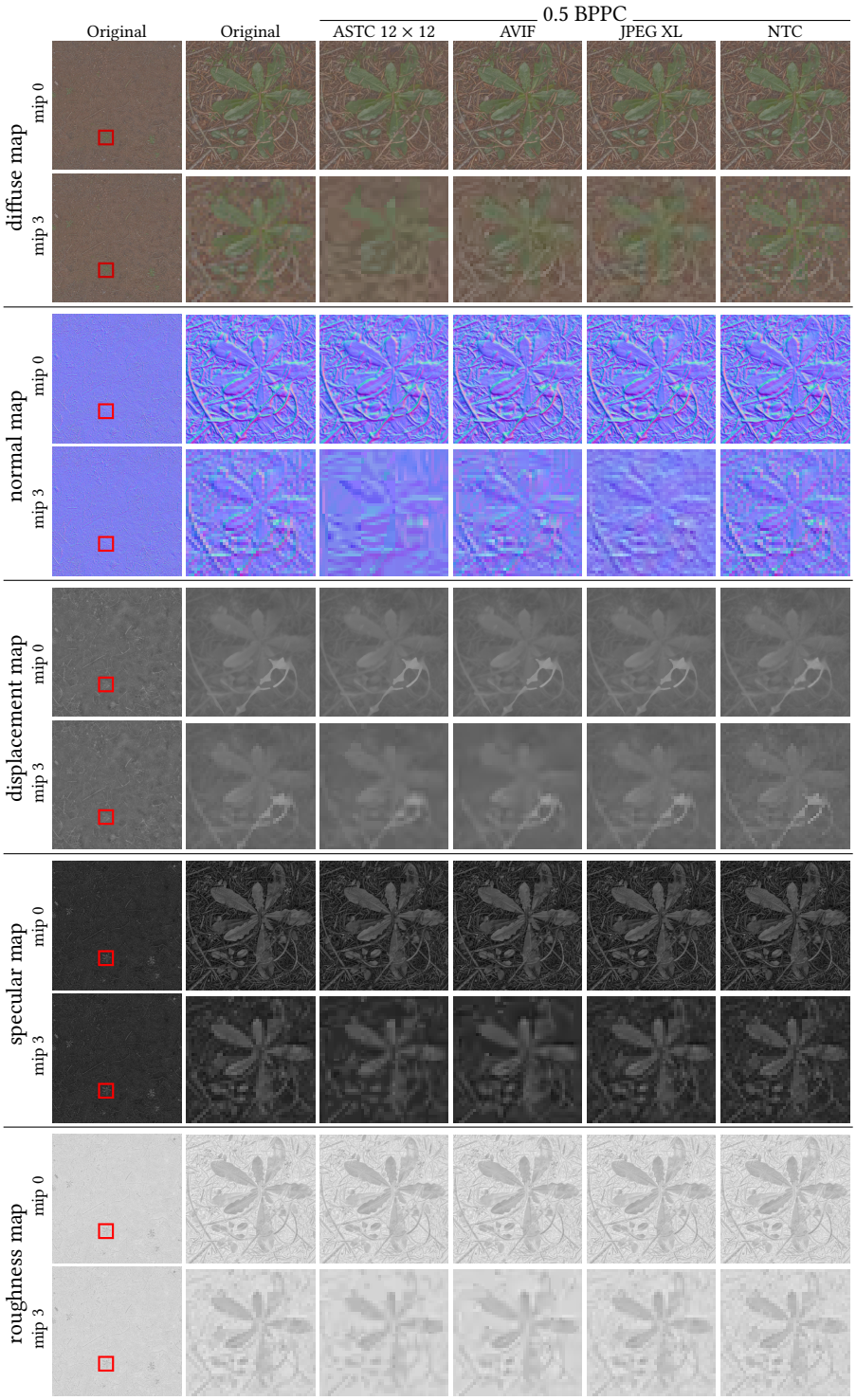

Fig. 17. Comparison of different methods at 0.5 BPPC, where we selected to show a texture set for which NTC's PSNR was close to its average PSNR over all texture sets in our 20 texture evaluation dataset. Recall that neither AVIF nor JPEG XL provide random access to the texture data. For visualization purposes, the diffuse images were exposure compensated with factor −1.0 and tone mapped with ACES [50]. Textures retrived from https://kaimoisch.com/free-textures/.

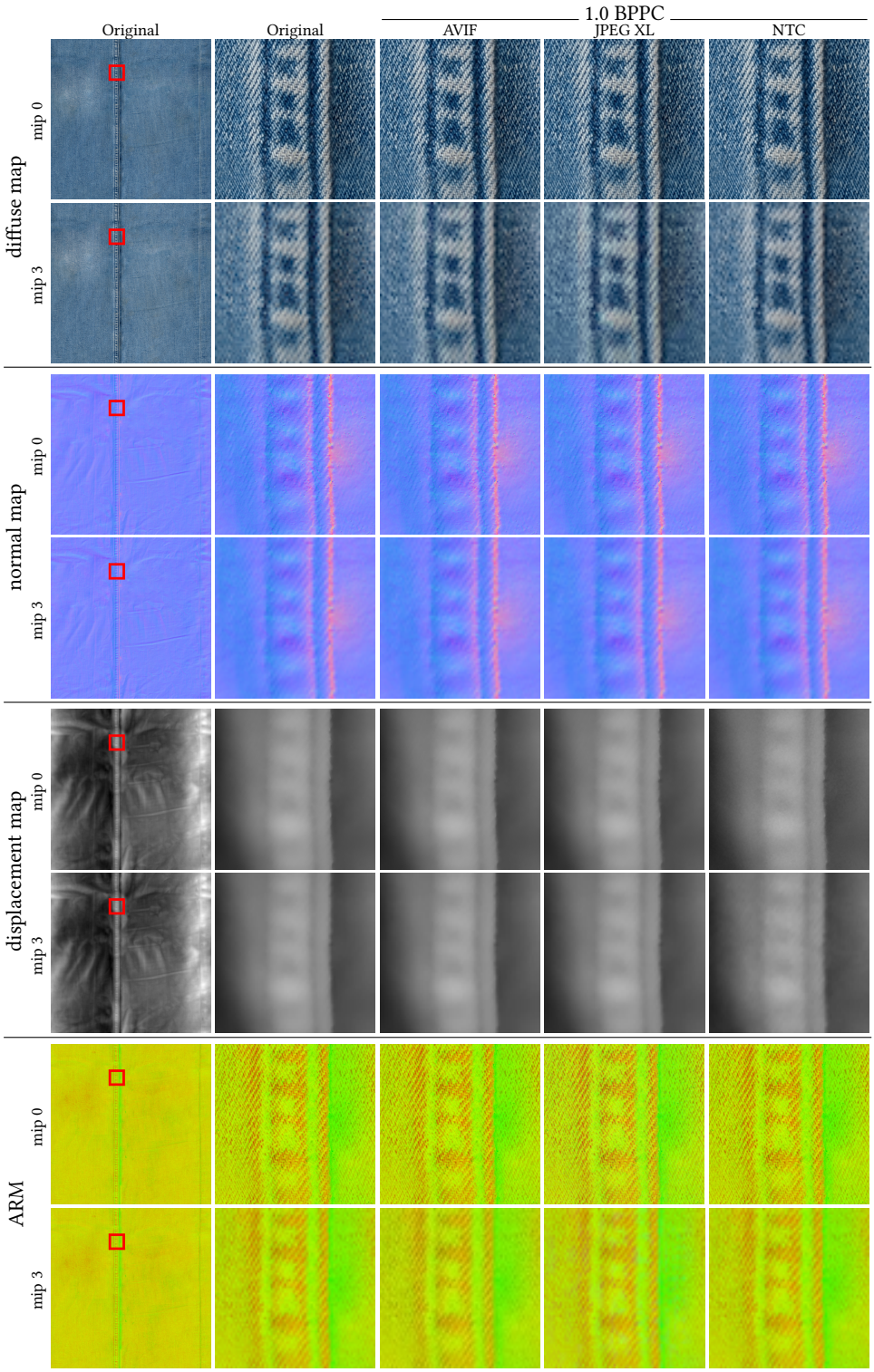

Fig. 18. Comparison of different methods at 1.0 BPPC, where we selected to show a texture set for which NTC's PSNR was close to its average PSNR over all texture sets in our 20 texture evaluation dataset. Recall that neither AVIF nor JPEG XL provide random access to the texture data. Textures retrived from https://polyhaven.com/.

F STORAGE COST

For completeness, we list the storage cost of NTC in Table 6 for our different compression profiles and for different texture set resolutions.

G FAILED EXPERIMENTS

In the course of developing our method, we evaluated a few alternative methods for neural compression. We found that these methods, which are characterized by either increased complexity or inferior quality, were unsuitable for the task of texture compression of material textures. We present these findings below.

Warped Grids. Prior work [41] proposes to warp volumes with a non-uniform transformation for better resource utilization compared to a uniform grid. We hypothesized that a similar approach, applied to images, could achieve some of the benefits of nonuniform bit allocation of entropy coding. We found that the inclusion of warping grids led to an increase in PSNR scores between 0.1 and 0.9 dB, depending on the scale of the warping grid used. However, after compressing and quantizing the warp grid, all the observed benefits could be achieved by simply allocating similar additional amount of storage to our latent grids.

Nonuniform Quantization. We empirically observed that, prior to quantization, the distribution of our grid values closely resembles a truncated normal distribution. We tried adopting a normally distributed quantization scheme, but did not observe any quality improvement. We attribute this outcome to the fine-tuning of network weights after the freezing of the latent grids, which might compensate for the sub-optimal quantization distribution (see Section 4.2).

H USAGE OF OTHER COMPRESSORS

In this section, we describe which compressors we compare to and their parameters. Note that BCx and ASTC are specifically targeting texture compression/decompression on GPUs and are designed to be random-access without entropy encoding. JPEG XL and AVIF are more traditional image compressors and include entropy encoding, which is a set of techniques that do not mesh well with the random access requirement for textures. We have included them still, since they are industry standards and because it may be worthwhile to investigate how our method fares against such advanced techniques. In fairness, it should be noted that neither JPEG XL nor AVIF were likely designed to reach bitrates as low as 0.2 BPPC, which is NTC's lowest target. We have also used Basis/KTX2 [30], which is part of the Khronos standard. This format also uses entropy encoding, but during decompression, it can transcode to many existing block-based texture compressions schemes, e.g., BCx, ETC, ASTC.

H.1 BCx Compression

For BCx compression [45], we performed a smaller investigation of existing tools, including AMD's Compressonator,[1] NVIDIA's Texture Tools,[2] and Intel's Fast ISPC Texture Compressor.[3] We used eight diffuse textures, eight normal maps, seven displacement maps, and seven roughness textures for this evaluation. The diffuse, normal, and displacement maps were from PolyHaven, and the roughness textures from ambientCG. The average PSNR for these three compressors over all textures were within ±0.2 dB. While the ISPC texture compressor had the highest average score, we could not find a command line tool version of it, and compressing a large number of high-resolution texture sets, including mipmaps, manually with their GUI was was prohibitively expensive.

For one-channel textures, e.g., roughness and displacement maps, AMD's tool wrote incorrect output files, so we always used NVIDIA's tool for those. For the diffuse textures and normal maps, AMD's tool produced slightly better result, so we used AMD's tool for those. The highest parameter setting was used for NVIDIA's tool, while we used two refine steps for BC1 (AMD) and quality 0.25 for BC7 (AMD). Going above those settings, mostly increased compression times but not quality.

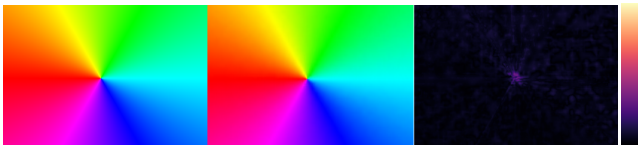

Fig. 19. A colorful and (presumably) difficult gradient texture compressed with NTC 0.2 does not show visible banding, color posterization, or discoloration. **Left:** Reference. **Middle:** Compressed. **Right:** ꟻLIP error image and corresponding color map.

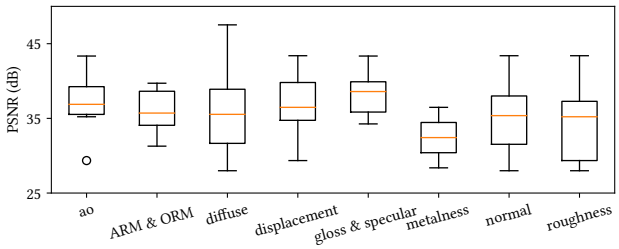

Fig. 20. Compression quality of different material properties for NTC 0.2. The orange lines show the median PSNR value for the respective texture types.

H.2 ASTC Compression

For ASTC [55], we used the texture tool from ARM, who developed ASTC,[4] with the `-exhaustive` flag, which provided best quality. Note that we used the two most aggressive variants of ASTC, which compressed 12×12 and 10×10 tiles. All variants store 16 bytes per tile, so using 12×12 tiles gives $128 \cdot 8/(12 \cdot 12) \approx 0.89$ bits per pixel for a three-channel texture. Furthermore, note that, in Figure 9 in the main paper, the ASTC results show average BPPC of around 0.5. This is a consequence of storing BPPC over an entire texture set, which may include both one- and three-channel textures. The same holds for other methods.

[1] https://gpuopen.com/compressonator/
[2] https://developer.nvidia.com/nvidia-texture-tools-exporter
[3] https://github.com/GameTechDev/ISPCTextureCompressor
[4] https://github.com/ARM-software/astc-encoder

Table 6. Storage cost of NTC textures based on the compression profile and texture resolution. The network parameter size depends on the compression profile, but is constant for different texture sizes. Storage cost is independent of the input channel count.

	NTC 0.2			NTC 0.5			NTC 1.0			NTC 2.25		
Resolution	2k×2k	4k×4k	8k×8k	2k×2k	4k×4k	8k×8k	2k×2k	4k×4k	8k×8k	2k×2k	4k×4k	8k×8k
NW (kB)	24	24	24	27	27	27	25	25	25	27	27	27
Grids (MB)	0.875	3.5	14.935	2.125	8.5	36.269	4.25	17.0	72.534	9.5	38.0	162.135
Total (MB)	0.899	3.524	14.959	2.152	8.527	36.296	4.275	17.025	72.559	9.527	38.027	162.162

Table 7. PSNR values with scalar quantization (SQ-8) and vector quantization (VQ-8, VQ-16) after optimization for 30k steps.

SQ-8	VQ-8	VQ-16
27.4 dB	27.28 dB	27.63 dB

H.3 JPEG XL

For JPEG XL [2], we used the reference implementation[5] and its precompiled executables (v0.8.0) from November 2022.[6] We started by performing lossless compression, and if that succeeded in reaching the target bitrate, our compression script exited. Otherwise, our script performed a binary search to find the quality setting that provided the sought-after bits per pixel per channel (BPPC). To reduce compression times, we did an early-out if the compression rate was within 2.5% of the target compression rate. We used the second highest value (8) for the effort parameter, since going to 9 (highest) provided little to no additional quality, but further increased compression times.

H.4 AVIF

For AVIF [13], we used precompiled executables using v0.11.1.[7] Similar to JPEG XL, our compression script for AVIF started by attempting to do lossless compression and exited if that reached the target compression rate. For all compression with AVIF, we used the "constant quality" `-a end-usage=q` flag, since this is common practice, and we also used quantization settings `-min 0 -max 63`. Next, our script performed a binary search on the `-a cq-level` quantization parameter *without* chroma subsampling. For very low bitrates, such 0.2 BPPC, this setting did not always reach the target. In those cases, our script continued with a new binary search with chroma subsampling enabled (`-yuv 420`). In the end, the file with the resulting bitrate closest to the target bitrate was selected. We used `-speed 3` since that resulted in reasonable compression times and going lower did not substantially improve image quality.

H.5 Basis/KTX2

For Basis,[8] we downloaded the code in early January 2023, and compiled it to use OpenCL for faster compression. The flags we use for compression are: `-opencl -ktx2 -uastc -uastc_rdo_l 1.0 -comp_level 6`, where 6 offers the best image quality and takes the longest to compress. All our results uses the file size of the output from the compressor. For image quality, however, we unpacked the basis file and had their decompressor transcode them to BC4 and BC7, respectively, since those are the formats that we compare to in the main paper.

[5]https://github.com/libjxl/libjxl
[6]https://artifacts.lucaversari.it/libjxl/libjxl/latest/
[7]https://github.com/AOMediaCodec/libavif
[8]https://github.com/BinomialLLC/basis_universal

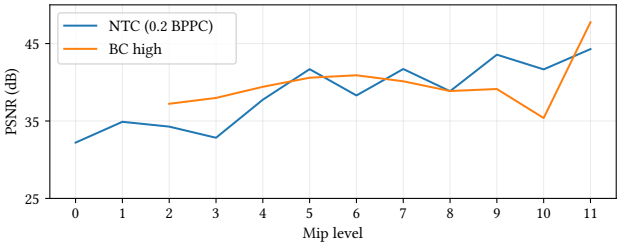

Fig. 21. Comparison of iso-storage compression quality at each mip level between our algorithm and the high-rate BC configuration, where we have used two fewer mipmap levels (0 and 1) to reach the same storage cost.

I ERROR AND QUALITY METRICS ON TEXTURE SETS AND MIP CHAINS

For reproducibility and future research, this section documents how our quality and error metrics were computed for texture sets and mip chains.

Given is a compressed texture set (including mip chains) $\mathbf{T} = \{\mathbf{T}_0^0, \mathbf{T}_0^1, \mathbf{T}_0^2, \ldots, \mathbf{T}_0^{M-1}, \ldots, \mathbf{T}_{N-1}^0, \mathbf{T}_{N-1}^1, \mathbf{T}_{N-1}^2, \ldots, \mathbf{T}_{N-1}^{M-1}\}$ with N textures and M mip levels, where mip j of texture i has resolution $w_j \times h_j \times c_i$. The values in the textures are assumed to be in $[0, 1]$. For each mip level, j, we concatenate the corresponding textures, creating a tensor $\mathbf{T}^j$ of shape $w_j \times h_j \times c$, where $c = \sum_i c_i$. The same concatenation is done for the reference texture set, $\mathbf{R}$, which contains the same number of textures as $\mathbf{T}$.

We compute the peak signal-to-noise ratio (PSNR) of $\mathbf{T}$ by summing the squared error over the mip chain and dividing by the total number of values in the tensor, yielding its mean squared error (MSE). That is, the PSNR is computed as

$$\mathrm{PSNR}(\mathbf{R}, \mathbf{T}) = -10 \log \left(\frac{\sum_j (\mathbf{R}^j - \mathbf{T}^j)^2}{c \sum_j w_j h_j} \right). \tag{1}$$

Computing LPIPS [88] for a texture set is a slightly more involved operation, as it requires 3-channel input. We consider a texture $\mathbf{T}_i^j$. If it only has a single channel, we repeat its channels to give a 3-channel texture, $\tilde{\mathbf{T}}_i^j$. If $\mathbf{T}_i^j$ has three channels, we let $\tilde{\mathbf{T}}_i^j = \mathbf{T}_i^j$. We then normalize the texture so that its values are in $[-1, 1]$, as is required by LPIPS, creating $\bar{\mathbf{T}} = 2\mathbf{T}_i^j - 1$. The same procedure is

used to create $\bar{\mathbf{R}}_i^j$. Next, we compute LPIPS between the two tensors $\bar{\mathbf{T}}_i^j$ and $\bar{\mathbf{R}}_i^j$. LPIPS yields an aggregate number $l_i^j = \text{LPIPS}\left(\hat{\mathbf{R}}_i^j, \hat{\mathbf{T}}_i^j\right)$ for the texture. We multiply the result by the number of channels c_i, effectively making a 3-channel texture worth three times as much as a single-channel texture when computing LPIPS over the texture set. Note that this is similar to the PSNR computations above. In addition, we multiply the result by the number of texels in the texture. We again sum the results over all levels of the mip chain and divide by the total number of values in the texture set. Formally, we have

$$\text{LPIPS}(\mathbf{R}, \mathbf{T}) = \frac{\sum_i \sum_j w_j h_j c_i l_i^j}{c \sum_j w_j h_j}. \tag{2}$$

We note that LPIPS was computed using the `net='alex'` LPIPS model, as proposed by the authors of the metric. Furthermore, LPIPS requires images that have resolutions of at least 32×32. For textures smaller than this, we apply zero-padding to make their sizes 32×32.

The structural similarity index (SSIM) [81] is a single-channel measure, providing a quality index between 0 and 1, where higher is better. To instead report errors, we compute 1−SSIM. We get an SSIM value for each channel in each texture in the mip chain. Similarly to the other two metrics, we weigh the result by the number of texels on the current mip level and normalize by the number of values in the texture set to get the final result. We get

$$1 - \text{SSIM}(\mathbf{R}, \mathbf{T}) = 1 - \frac{\sum_i \sum_j w_j h_j c_i \text{SSIM}\left(\mathbf{R}_i^j, \mathbf{T}_i^j\right)}{c \sum_j w_j h_j}, \tag{3}$$

where $\text{SSIM}\left(\mathbf{R}_i^j, \mathbf{T}_i^j\right)$ computes SSIM for each channel in the texture and returns the average. The SSIM computations include filtering with an 11×11 Gaussian kernel. We do not apply the kernel for images that are smaller than the kernel.

When we report LPIPS and SSIM errors over the entire data set, we take the mean of the errors computed for each texture set in the data set. For PSNR, we compute the average of the per-texture MSE values retrieved during the computation of the PSNR values. The aggregate PSNR is then computed using that average. Note that neither of these three aggregate error and quality values consider the resolution of the textures in the texture set nor the total number of channels present within it. The effect of this is that each texture in our diverse texture set has the same impact on the aggregate score. Furthermore, note that the computations in Equations 1-3 gives more weight to the lower levels (i.e., higher resolutions) of the mip pyramid compared to the higher. The reasoning behind this is that the lower levels will cover more pixels when used in rendered images.

Finally, we note that when we, for Figure 21, compute the average quality for mip level m over the entire data set, we aggregate over all available mips at that level, independently of their resolution. Consider the aggregate error value for the second mip level, for example. Despite the second mip level of a texture with resolution 1024×1024 having resolution 512×512 while the second mip level of a 4096×4096 texture has resolution 2048×2048, we add both of their error values into the aggregate for the second mip level, without weighting based on resolution.

J EVALUATION TEXTURE SET DETAILS

Figures 22-24 show the twenty texture sets included in our evaluation set. The sizes of the textures range from 2048×2048 to 8192×8192. The textures were either created by the authors or retrieved from in-house or external sources. The external sources were: ambientCG (https://ambientcg.com/), EISKO© (https://www.eisko.com/), KaiMoisch (https://kaimoisch.com/free-textures/), and PolyHaven (https://polyhaven.com/). Consisting of only a diffuse texture, the "gradient 4k" texture set contains the fewest number of channels (three). This texture set is the only one created by the authors and is provided as a difficult case. The "Louise 4k" set contains the most channels (12 in total, consisting of diffuse: 3, normal: 3, roughness: 1, subsurface: 1, ambient occlusion: 1, displacement: 1, gloss: 1, and specular: 1). Grayscale textures that were stored as RGB were converted to one-channel textures and constant textures were removed. EISKO and KaiMoisch provided 16-bit textures, which we converted to 8-bit before we used them.

The normal maps were originally provided as three-channel textures. We note that the BC5 format is a two-channel format targeting normal maps, under the assumption that the third component can be computed if the normals are of unit length. Intially, we planned to convert all normal maps to two channels, but found that the majority of the normal maps did not have normals of unit length. Since the length of the normal sometimes is used to store another property (for filtering normals, for example), we left them as three-channel textures to avoid changing the intent of the texture artists. However, since the quality of the normals is often extremely important, we used the higher-quality BC7 format (instead of BC1).

We create the mipmap chains with a high-quality Lanczos downsampling filter and stop when we have reached the level with a resolution of 4×4 pixels, since that is the tile size of most block-based GPU compression schemes.

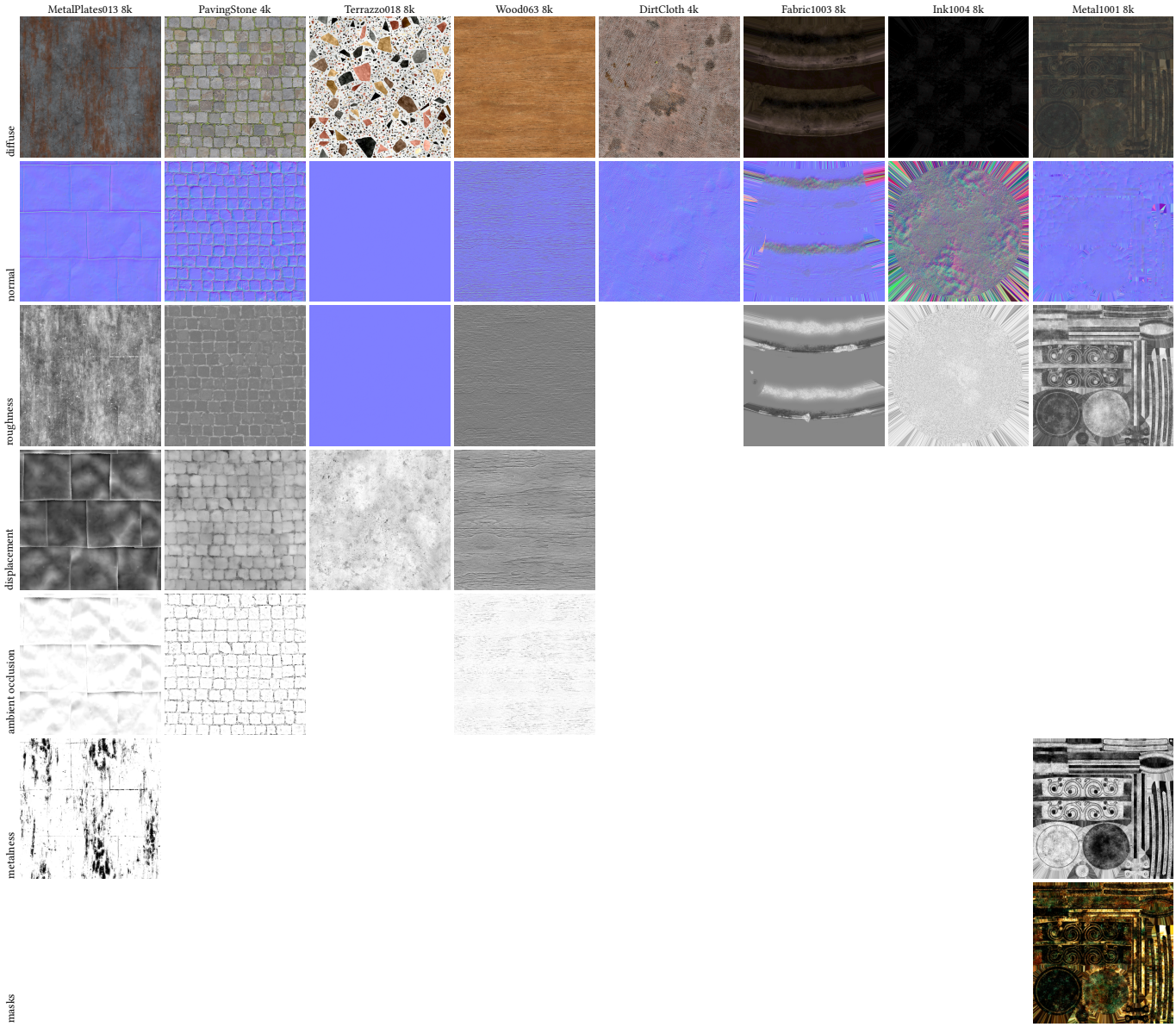

Fig. 22. Texture sets number 1.

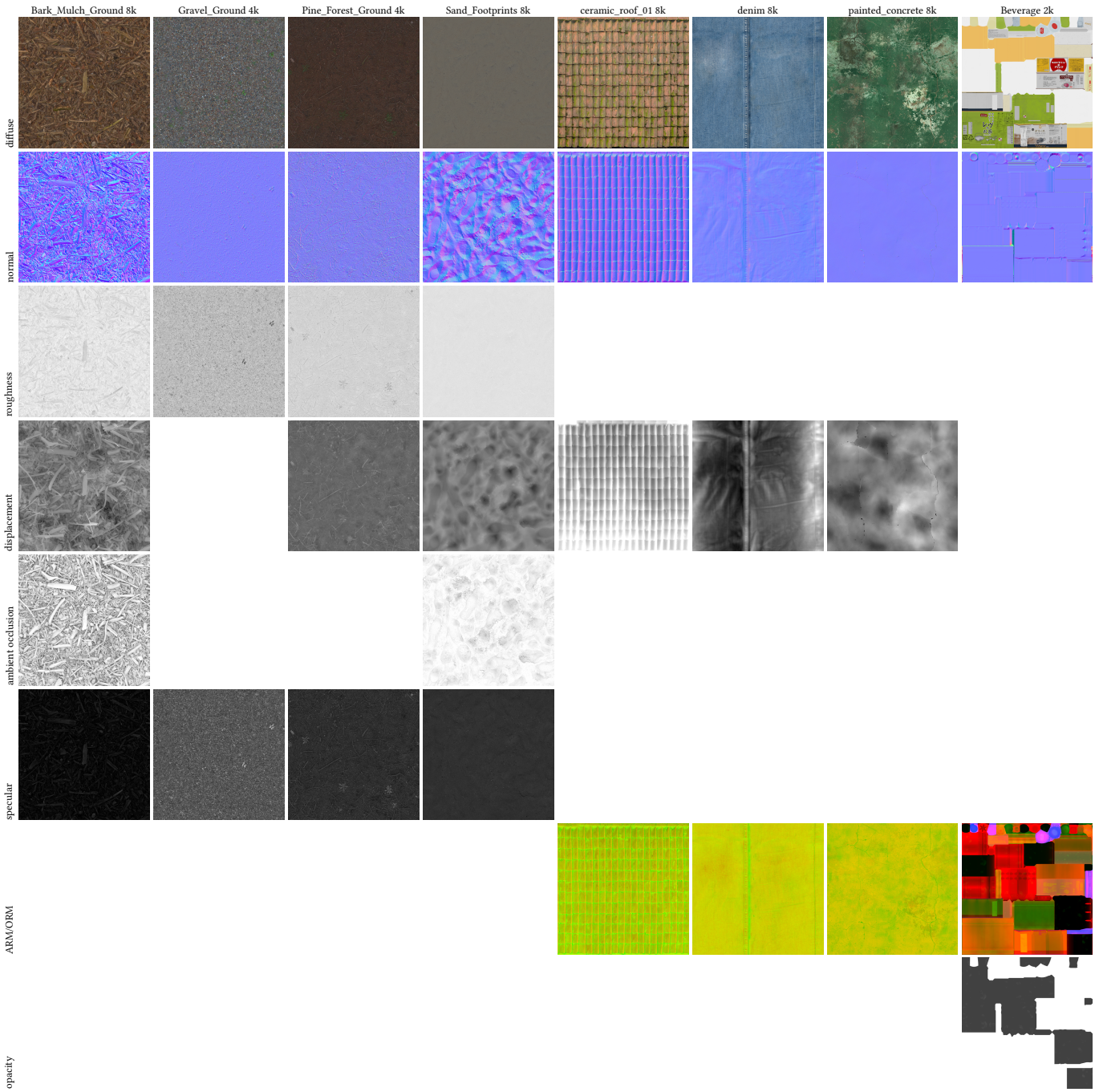

Fig. 23. Texture sets number 2.

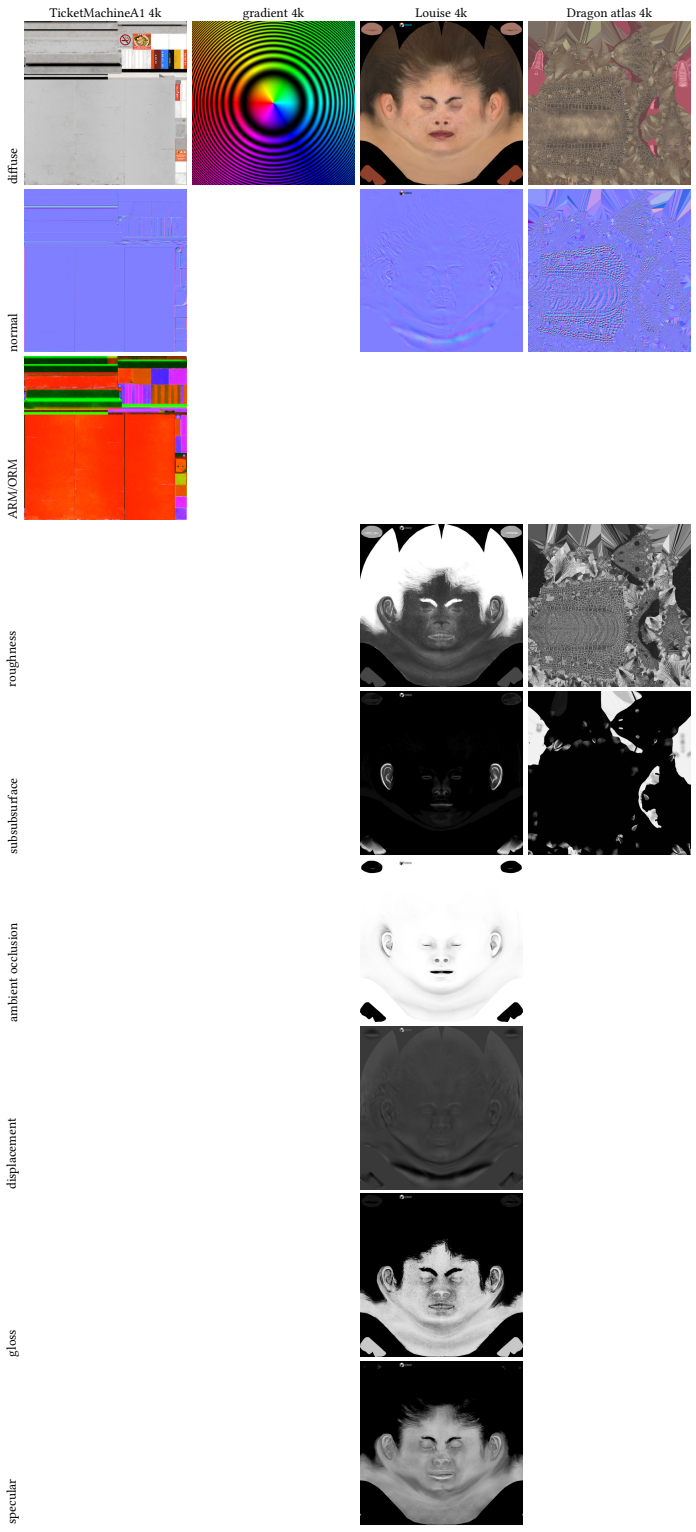

Fig. 24. Texture sets number 3. Animatable Digital Double of Louise by EISKO© (www.eisko.com).