Supplementary Materials for TileGen: Tileable, Controllable Material Generation and Capture

Xilong Zhou
Texas A&M University
College Station, USA

Miloš Hašan
Adobe Research
San Jose, USA

Valentin Deschaintre
Adobe Research
London, UK

Paul Guerrero
Adobe Research
London, UK

Kalyan Sunkavalli
Adobe Research
San Jose, USA

Nima Khademi Kalantari
Texas A&M University
College Station, USA

ACM Reference Format:
Xilong Zhou, Miloš Hašan, Valentin Deschaintre, Paul Guerrero, Kalyan Sunkavalli, and Nima Khademi Kalantari. 2022. Supplementary Materials for TileGen: Tileable, Controllable Material Generation and Capture. In *SIGGRAPH Asia 2022 Conference Papers (SA '22 Conference Papers), December 6–9, 2022, Daegu, Republic of Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3550469.3555403

Additional implementation details. In this section, we discuss network architecture and dataset generation details.

As discussed in Section 3.2 and Fig. 2 of the main paper, TileGen follows a CollageGAN-based encoder and StyleGAN2-based decoder architecture, with the difference that we use wrap-around versions of all convolution and upsampling operations. Please refer to CollageGAN and StyleGAN2 about architecture details; here we only discuss how we realize the wrap-around operation. There are two operations that need to be replaced: regular convolution and upsampling (achieved by transposed convolution). For convolution, we apply the "circular" padding mode in the regular Pytorch 2D convolution operation. The upsampling operation in original StyleGAN2 is achieved by transposed convolutions; instead, we use wrap-around upsampling followed by wrap-around convolution. More specifically, to achieve wrap-around upsampling for input of size $h \times h$, we manually add circular padding to the input, to get a padded input of size $(h+1) \times (h+1)$, followed by bilinear interpolation to resample the padded input from $(h+1) \times (h+1)$ to $(2h+1) \times (2h+1)$. Then we crop the middle $2h \times 2h$. The bilinear interpolation ensures the corners of the original tile map to the new tile corners. Next, we apply a standard learnable circular convolution to generate a tileable output of size $2h \times 2h$. A correct implementation ensures that the generator always produces tileable results, including even before training.

Next, we illustrate how the dataset is generated. As is discussed in Section 3.3, Substance Source and the default class tags are used to generate a class-specific dataset. For example, to prepare a tile/brick dataset, we collect a large number of Substance graphs tagged with keywords "tile" or "brick". Note that Substance Source website categorizes and searches many materials into groups and subgroups based on the artistic provided tags However, the accuracy of this

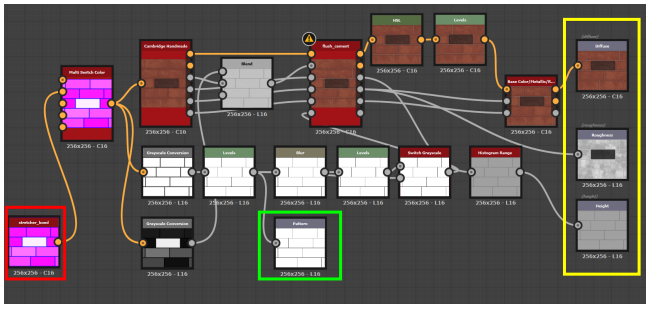

Figure S1: This figure illustrates how we pair conditional patterns and material maps using a simplified tile graph example. The pattern generator node (marked as red) is processed by several nodes to generate a reasonable pattern, where we insert a export node (marked as green) to export conditional maps. In this example, when the parameters of the graph are augmented to produce varying material maps (marked as yellow), the condition map will also vary and the map will always be consistent with the generated material maps. Note that even though graphs can vary substantially, we are able to follow the same logic to produce condition maps.

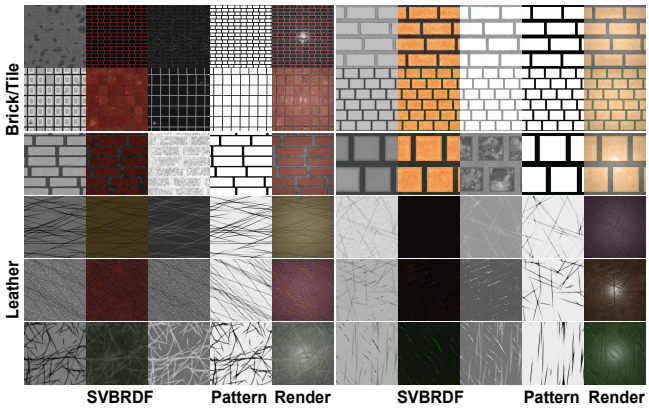

Figure S2: Examples of tile and leather dataset. We show height, diffuse, roughness, and condition maps, as well as the rendering of each material.

SA '22 Conference Papers, December 6–9, 2022, Daegu, Republic of Korea

artistic-provided material definition has not been fully studied and the sensitivity of material categorization on the results of TileGen is not covered in this paper. In the future, it would be interesting to explore the effect of material categorization on the material generative model.

Then we discussed how the condition patterns are generated. As shown in Figure S1, we determine the nodes related to the pattern generator (or the nodes that produce plausible patterns) in each graph and pipe it into a new export node. Since these pattern generators are part of the graph, the final generated materials maps are always consistent with the patterns. Next, the exposed parameters of the graphs can be augmented automatically to produce diverse results from each graph. In tiles and bricks, we reduce the range of parameters controlling diffuse color to avoid unrealistic colors. For each graph, we generate about 200 material maps; in total we have around 20k material maps for each class.

In the Figure S2, we demonstrate several examples of dataset of tile and leather. More specifically, we show the maps for height, diffuse, roughness, condition and the rendered image for each material. For tile/brick dataset, the conditions are binary images defining the pattern of material and for the leather dataset, the conditions are gray-scale images defining the wrinkles of the materials. All conditions are perfectly paired with the resulting maps. Note that the three top-right tile examples and three bottom-right leather examples are sampled from the same tile and leather graph respectively, demonstrating the variety achievable within a single graph. Note that our tile dataset does not contain rotated tile patterns or irregular patterns, though these could be added.

Additional results for generation and inverse rendering. In Figure S3 we show individual SVBRDF maps for the conditionally generated results shown in Figure 3 of the main paper. More conditionally generated results are shown in Figure S4. The SVBRDF maps for the unconditional results in Figure 4 of the main paper are shown in S5, and more unconditional results in Figure S6. Figure S7 shows additional inverse rendering results.

Nearest neighbours. In Figure S8, we show randomly sampled stone materials and top three nearest-neighbour materials from the stone dataset, as measured by the distance of Gram matrices of VGG features. This shows that our model is not simply overfitting to the dataset.

Interpolation results. In Figure S9, we show latent-space interpolation results between two randomly sampled materials for leather, metal, stone and tile.

More comparisons. In Figure S10, we show a comparison of material acquisition with MaterialGAN [Guo et al. 2020b] and real images. As shown, our approach is able to reproduce re-renderings closer to the ground truth images. The material maps generated by our method are more consistent and lack artifacts compared to MaterialGAN.

Effect of input patterns on inverse rendering. We demonstrate the effect of conditional input patterns on the results of inverse rendering. In Figure S11, the top three rows are leather examples and bottom three rows are tile examples. For each example, we match the same target image with three different input patterns.

We can see that in the leather examples, all input patterns produce plausible reconstructions that both capture the style of target image, and also preserving the "wrinkles" specified in the input patterns. In the right-most leather example, the provided wrinkle patterns are significantly different from the wrinkles in the target image, but our result still capture the style of the target while using the wrinkles provided by the input patterns.

The bottom three rows show that for tile examples, the input patterns do not need to be exactly aligned with the target images (Different examples of Figure 5 also demonstrate this property). And between input patterns and target images there exists some "tolerance", including tile number, tile size, the thickness of grout and the offset of tiles. However, if the feature sizes of the input patterns are significantly different from the target images, our model will try to mix style and patterns and generate unrealistic bricks (see the 3rd patterns for each tile example). This is because the Gram matrix loss is shift-invariant but not scale-invariant, and significantly mismatched patterns will force a change in the style computed by the Gram matrix.

Results larger than inputs. Furthermore, we demonstrate that our generated materials can have higher resolution than the target image. In Figure S12, we use a 256×256 target photograph and provide a pattern for a 512×512 output domain, with a feature size matching the target image in pixel units. Our method results in tileable materials of extended size, which is not possible with previous pixel-based capture methods.

Translated condition with fixed style. In Figure S13, we translate the input pattern, while holding the latent code fixed. The resulting materials follow the shifted pattern but do not change their style, which further demonstrates that the style of the result is mainly derived from the latent vector rather than the pattern encoding. This property is desirable for inverse optimization, where the pattern is fixed and we would like to optimize for the material style.

Conditioning on colored patterns. In addition to conditioning on binary patterns, we can also train our generator to allow conditioning on colored patterns to control the diffuse albedo in the generated materials —other properties of other materials classes could be conditioned this way. In Figure S14, we provide a pattern with colors roughly matching the target photograph, but at different image locations. The generator produces a random style that matches the given colors. Optimization to match the target image further reproduces the detailed style of the target photograph, even though the colors in the pattern already provide significant prior information.

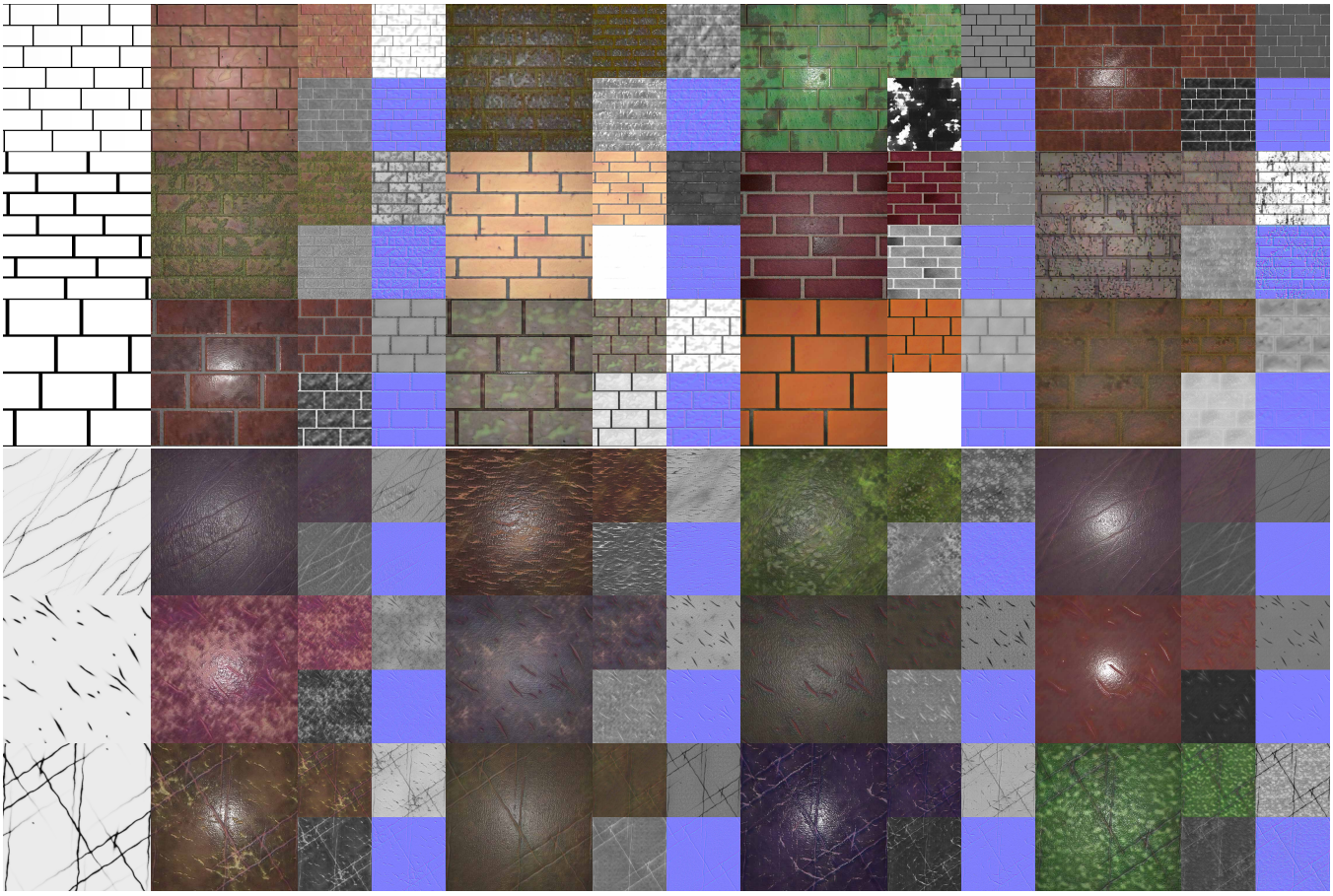

Figure S3: Randomly sampled conditional results with the generated texture maps of tile and leather examples in the Figure 3 of the paper.

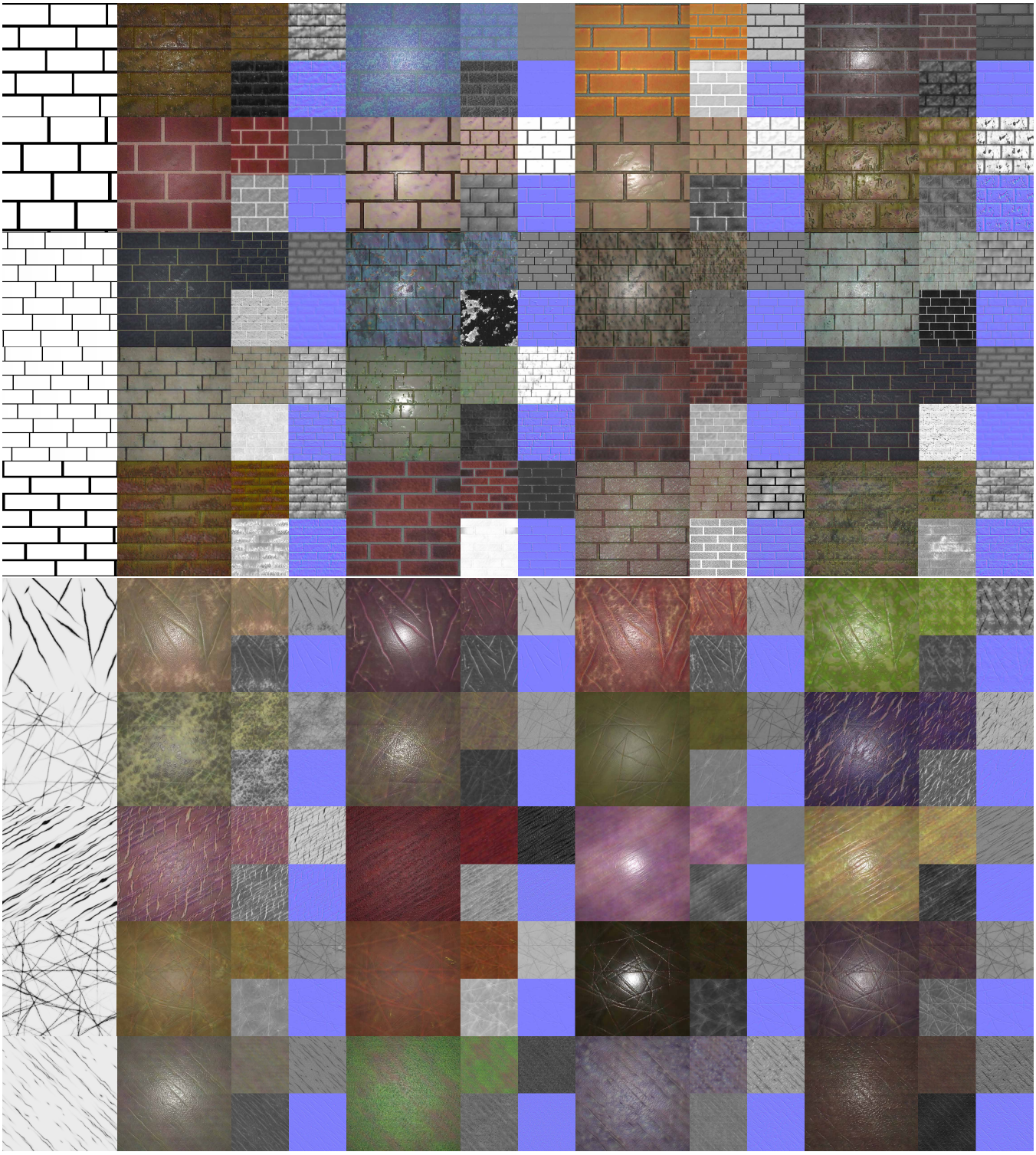

Figure S4: Additional randomly sampled conditional results with the generated texture maps of tile and leather material class.

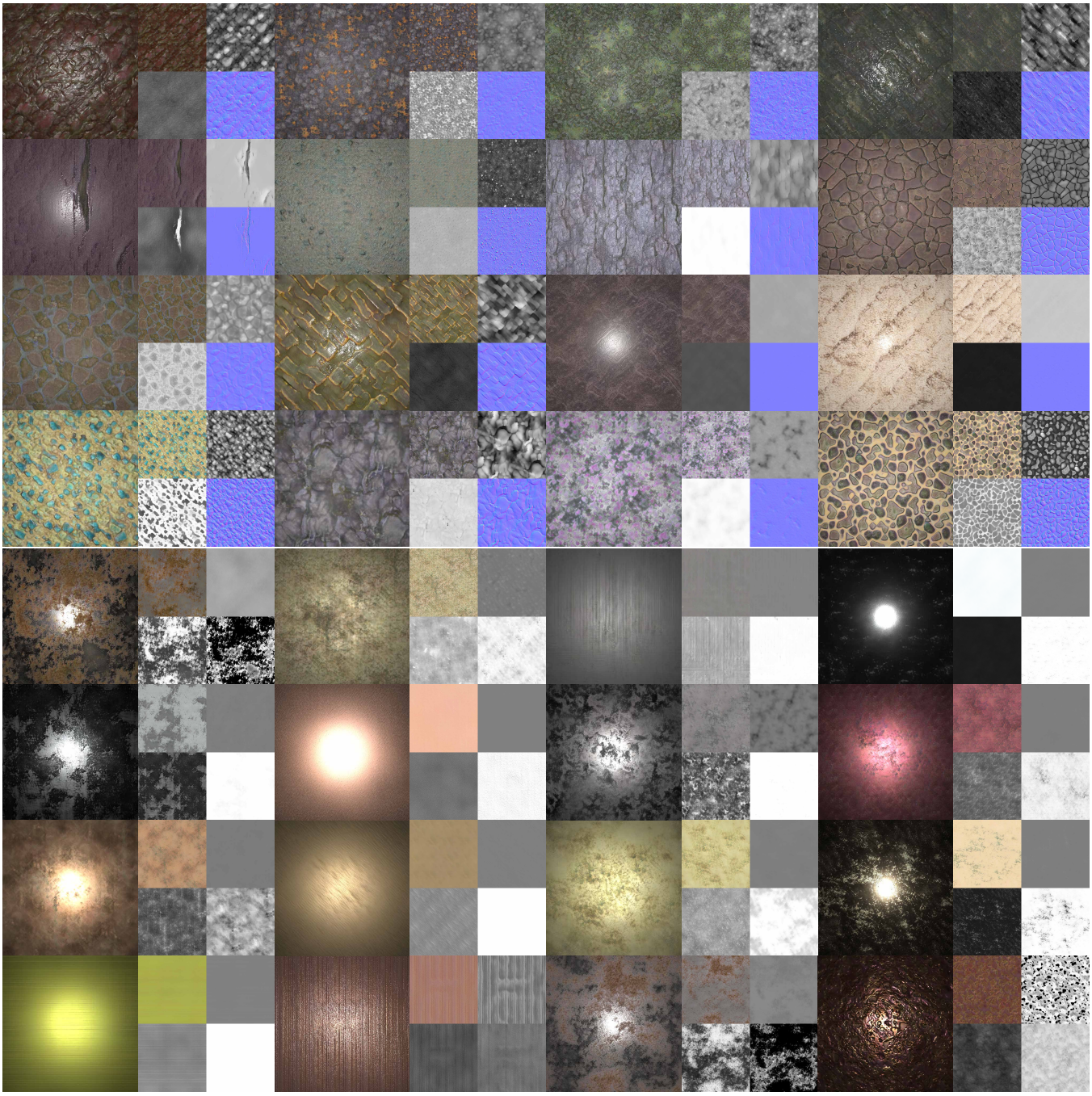

Figure S5: Randomly sampled unconditional results with the generated texture maps of stone and metal examples in the Figure 4 of the paper.

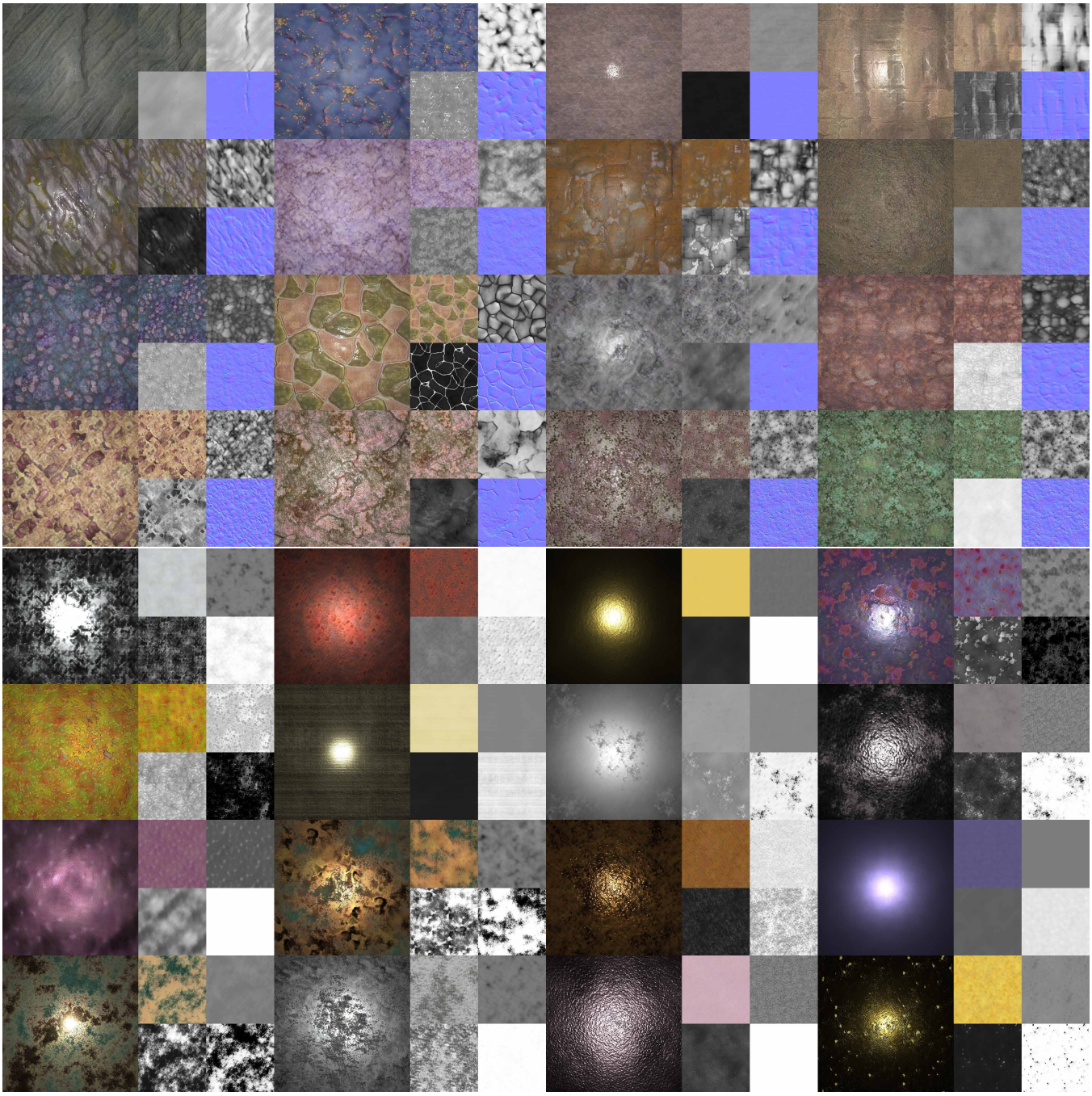

Figure S6: Additional randomly sampled unconditional results with the generated texture maps of stone and metal material class.

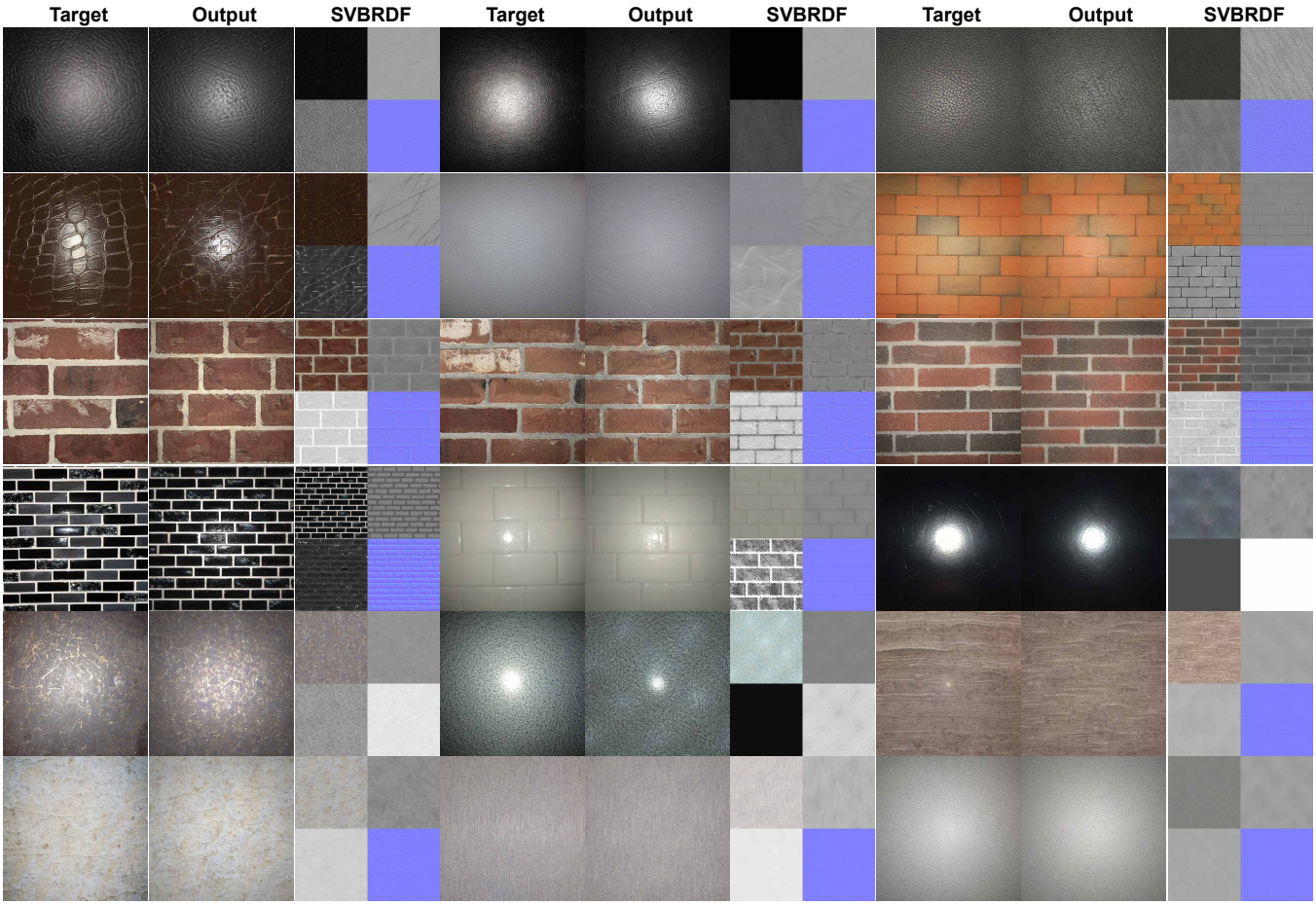

Figure S7: Additional inverse rendering results with the generated texture maps.

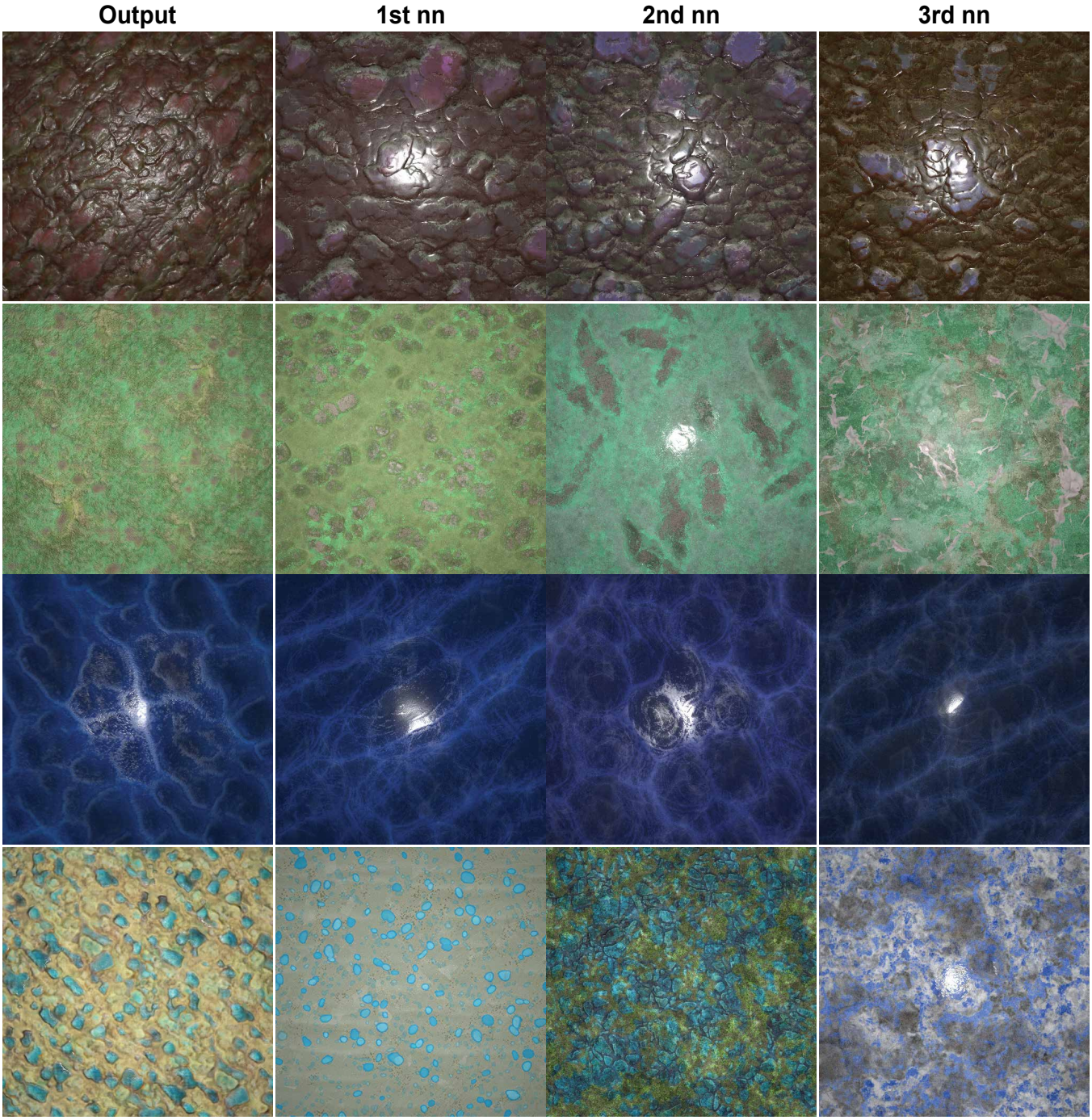

Figure S8: Nearest neighbours of sampled stone examples, measured by Gram matrix distance. We show sampled stone examples followed by top three nearest neighbour materials from dataset. The difference shows that our model is not simply overfitting to the dataset.

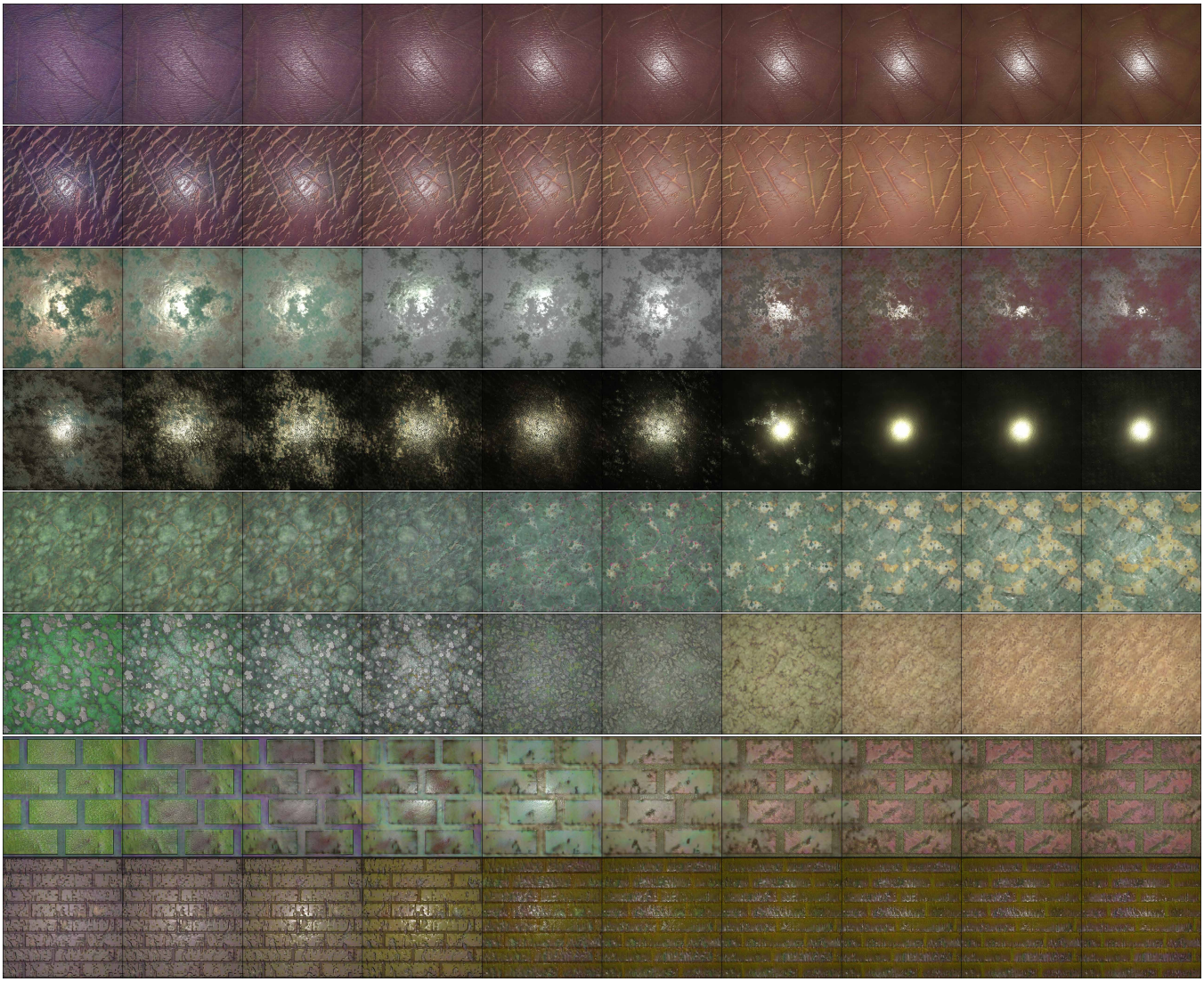

Figure S9: The results of style interpolation between two random sampled latent space for leather, metal, stone and tile.

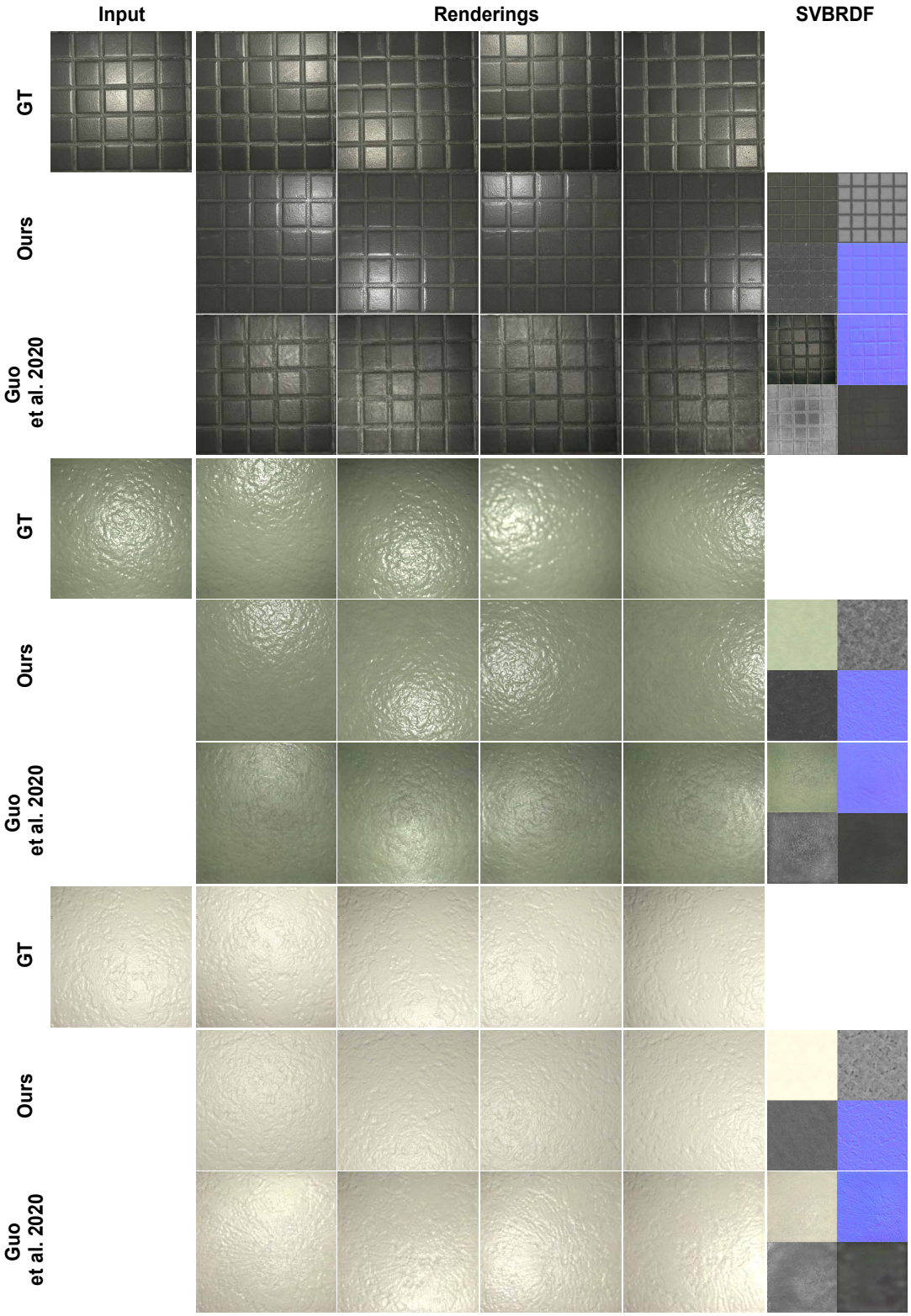

Figure S10: Comparison with MaterialGAN and real images. We show three reconstructed materials maps as well as re-renderings using our method and MaterialGAN. As is shown here, compared with real images, our method produces much plausible maps and renderings without overfitting artifacts.

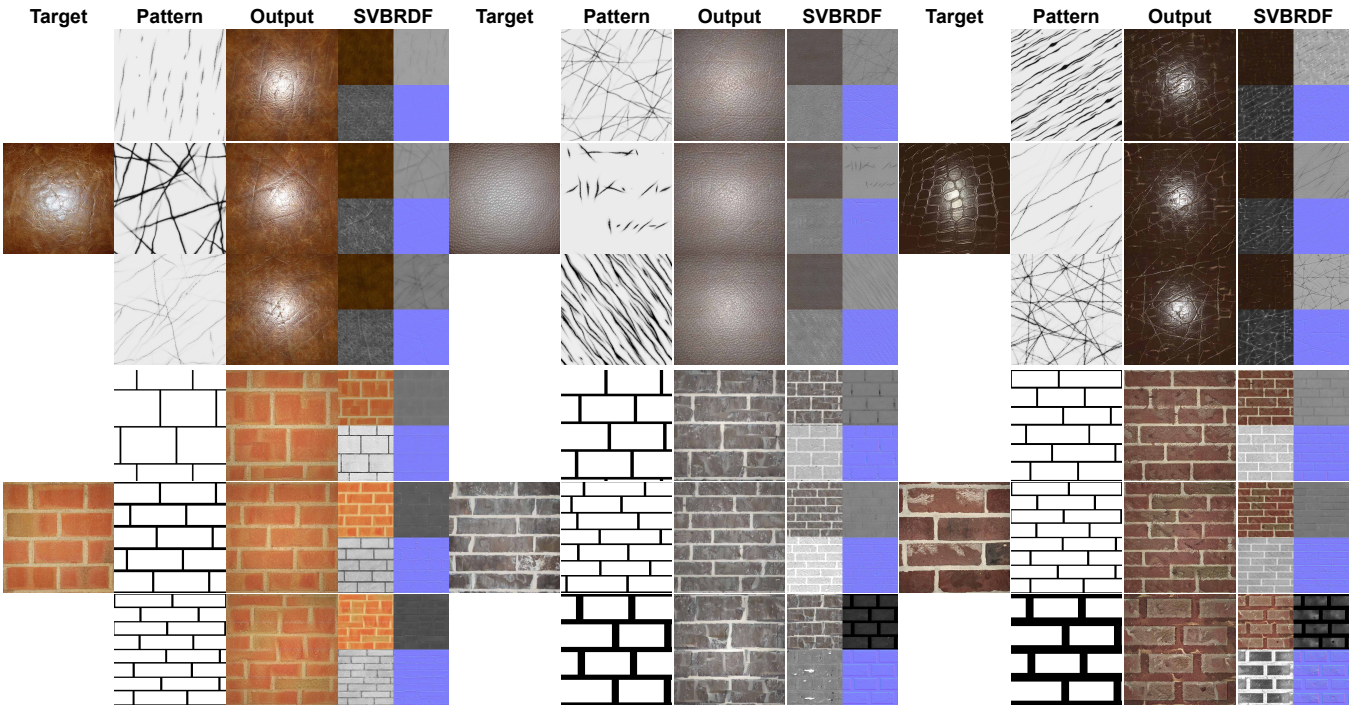

Figure S11: This figure shows the effect of conditional input patterns on the generated feature maps given a certain target image. For leather examples, our results can capture the style of target images and maintain "wrinkles" introduced from various input patterns; for tile examples, given input patterns that are not significantly misaligned with target images, our model can still generate realistic results.

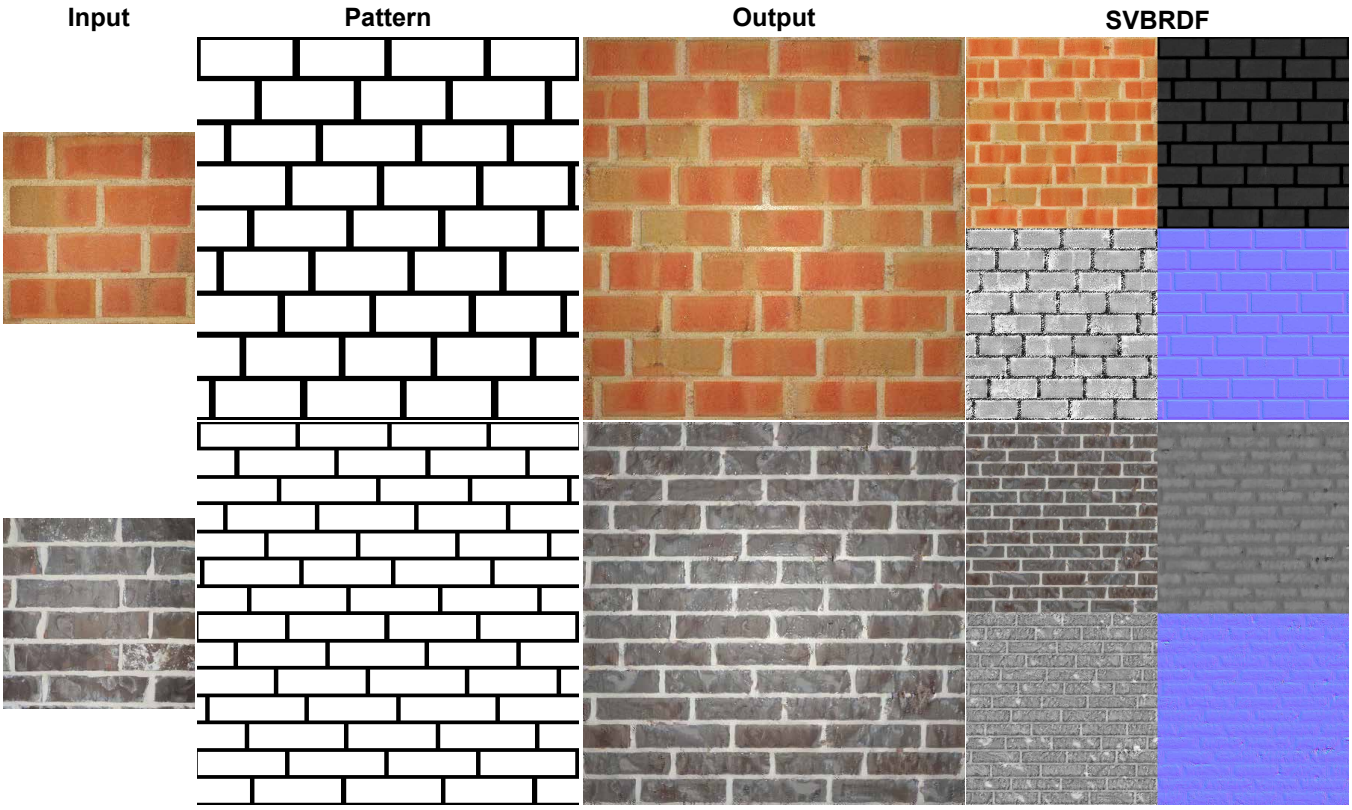

Figure S12: The textures resulting from our optimization can be larger than the target image. Here we start from 256×256 target images, and define a pattern with a feature size matching the target in pixels, but covering a 512×512 output domain. Our method is able to produce tileable materials of extended size, which is not possible with previous pixel-based methods.

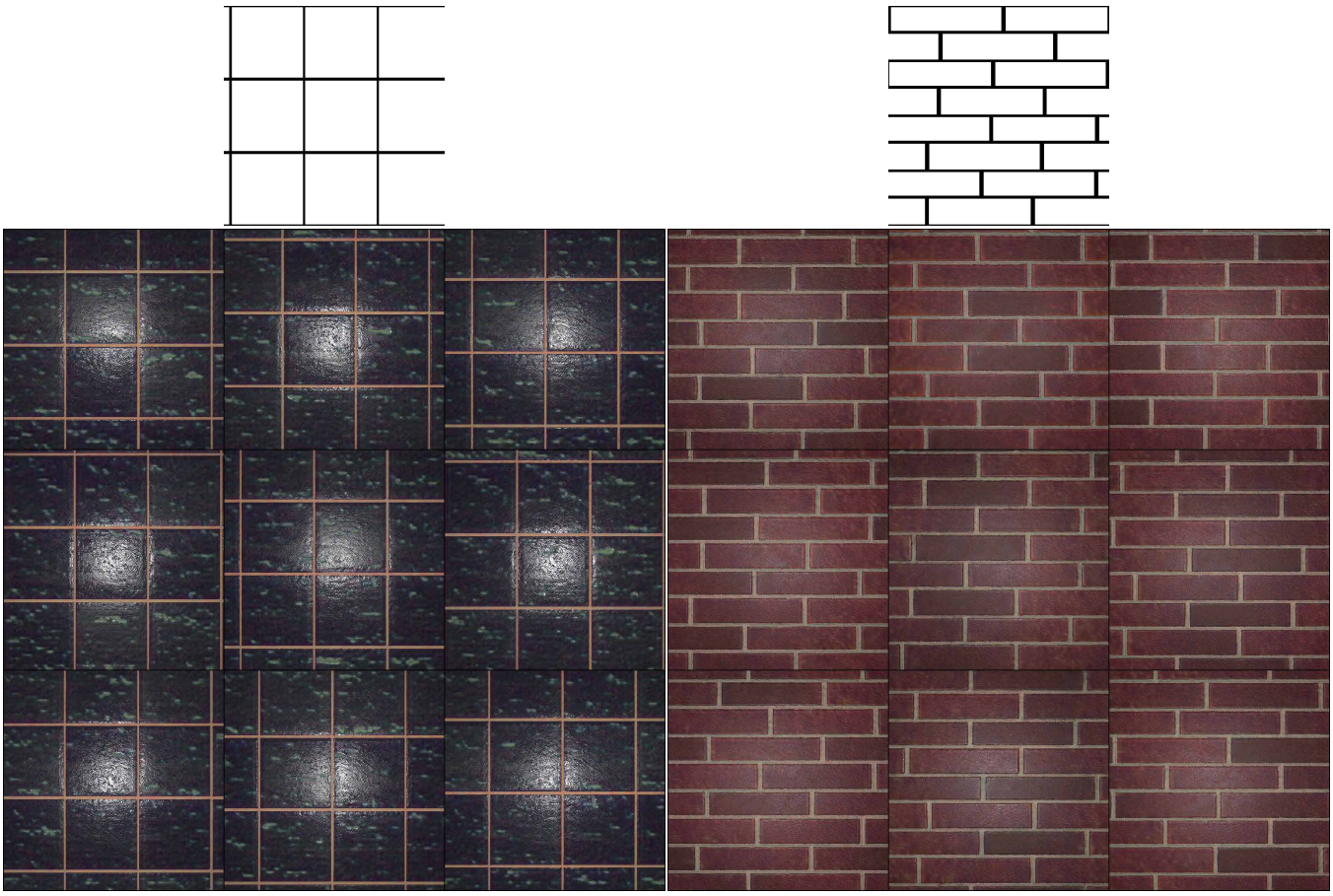

Figure S13: A fixed input pattern (shown on the top) is translated by a number of pixels, while keeping a fixed latent code. The resulting material follows the shifted pattern and keeps a similar style, showing that much of the style of the result is derived from the latent vector, not the pattern encoding. This shows that our model provides a certain amount of disentanglement between condition and style, which is sufficient for our forward and inverse tasks.

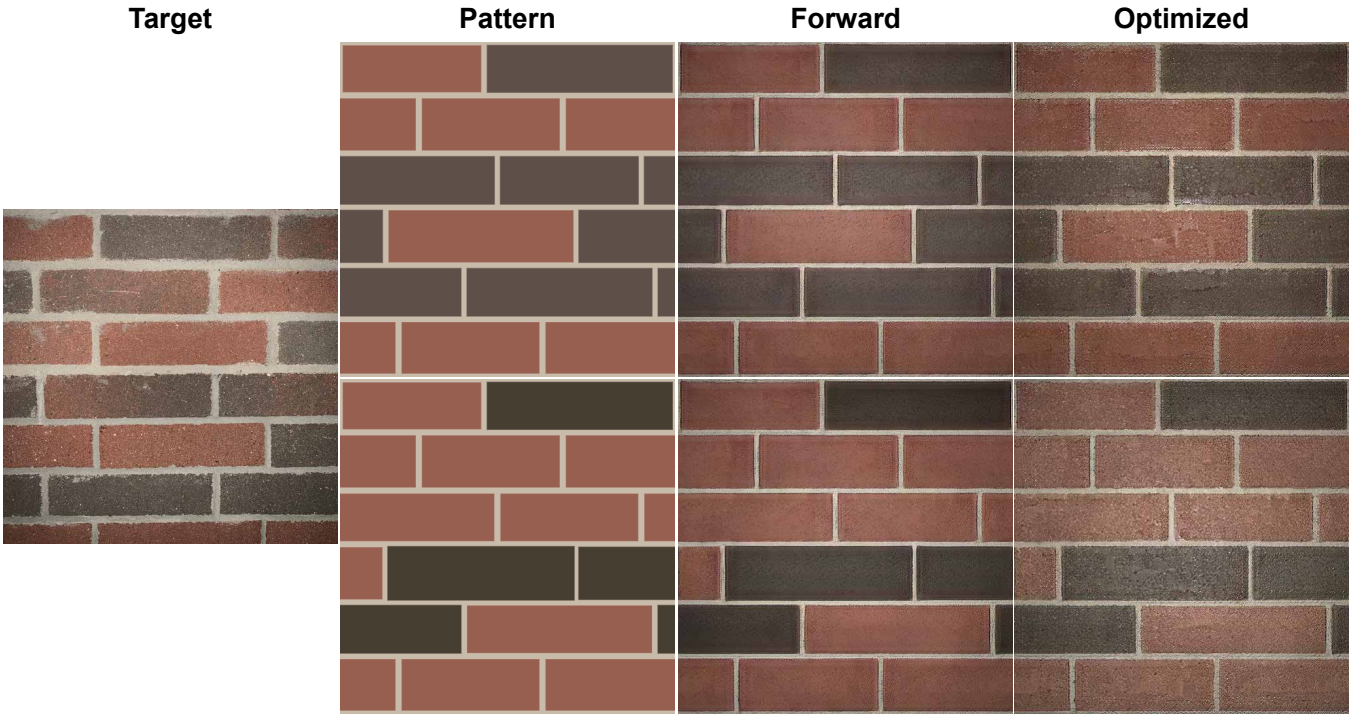

Figure S14: Here we demonstrate the color version of our conditional generator. We provide a pattern with roughly matching colors to the generator (forward) and optimizer (Optimized). Even though the colors provide significant information which doesn't align well with the input picture, the optimization is able to further match the style of the target photograph.