Machine Vision-Oriented Appearance Design: Generate Natural And Robust Textures For 3D Meshes

Supplementary Material

6. Structure of ADN

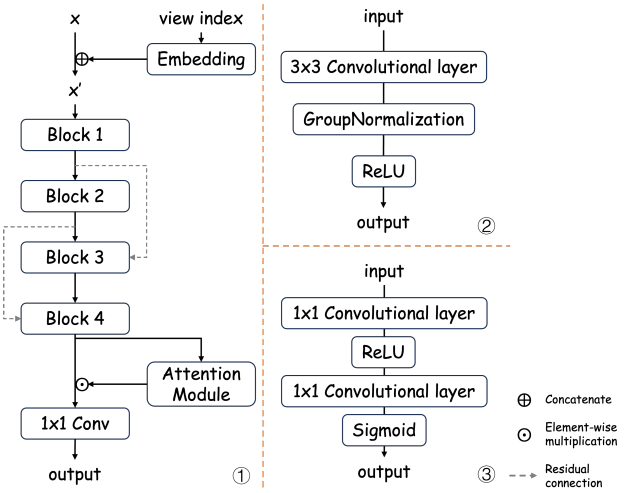

Figure 8. 1. Schematic of the ADN architecture. 2. Internal structure of the block. 3. Internal structure of the attention module.

First, we detail the architecture of the Aggregation Decision Network (ADN). It takes as input a tensor x of shape $(B, 4, H, W)$ and a tensor view index of shape $(B,)$. For the B views, the view index contains the camera index for each sample. The tensor x is defined as:

$$x = \text{Concat}(m, c, g, p), \tag{20}$$

where $m, c, g, p \in \mathbb{R}^{B\times 1\times H\times W}$ denote the visibility mask of triangles under the view, the cosine between the viewing direction and the surface normal, the gradient map, and the pixel-intensity map obtained after UV re-mapping, respectively. In the ADN, the view index is first mapped through an embedding layer and reshaped to match the spatial size (H, W) of x. The concatenated tensor x' is then processed by four convolutional blocks; after the last block, a lightweight attention module is applied, and the final output is produced by a 1×1 convolution. Fig. 8 illustrates the ADN as well as the structural details of the convolutional blocks and the attention module.

7. Definition of Corruption Error(CE)

According to the definitions in [13, 31], corruption error (CE) is proposed to comprehensively evaluate a classifier's robustness to a given type of corruption. The first evaluation step is to take a trained classifier f, which has not been trained on IMAGENET-C, and compute the clean dataset top-1 error rate. Denote this error rate E^f_{clean}. The second step is to test the classifier on each corruption type c at each level of severity s $(1 \le s \le 5)$. This top-1 error is written $E^f_{s,c}$. Before aggregating the classifier's performance across severities and corruption types, error rates should be made more comparable since different corruptions pose different levels of difficulty. We adjust for the varying difficulties by dividing by the errors of target classifier (*i.e.* when using VGG16 for testing, we use VGG16's errors). So Corruption Error is computed with the formula:

$$\text{CE}^f_c = \Big(\sum_{s=1}^{5} E^f_{s,c}\Big) / \Big(\sum_{s=1}^{5} E^{\text{target}}_{s,c}\Big) \tag{21}$$

The **mean corruption error** (or mCE for short) can be calculated by averaging the 15 Corruption Error values. The authors further proposed a metric to indicate the amount that a classifier declines on corrupted inputs, which named Relative Corruption Error:

$$\text{Relative CE}^f_c = (\sum_{s=1}^{5} E^f_{s,c} - E^f_{\text{clean}}) / (\sum_{s=1}^{5} E^{\text{target}}_{s,c} - E^{\text{target}}_{\text{clean}}) \tag{22}$$

Averaging these 15 Relative Corruption Errors results in the **relative mean corruption error** (R.mCE). This measures the relative robustness or the performance degradation when encountering corruption.

8. More Visualization Results

Here we present additional visual results. Recognizing that different rendering engines may introduce subtle discrepancies in the rendered views, we examine the robustness of appearances generated with PyTorch3D across different renderers. Keeping the camera viewpoints fixed, we re-render the generated textures in MeshLab and Blender, and evaluate how predictions change under the same corruption types and severities. A subset of the results is shown in Fig. 9. As we can see, the appearances generated by our method remain reasonably robust across different rendering engines, although the confidence may decrease in some cases. By comparing the outputs of the different renderers, we can also observe that the appearances rendered by PyTorch3D are highly consistent with those rendered by MeshLab. This is because both are based on relatively simple real-time rasterization pipelines with simple shading models, whereas Blender employs a more complex rendering pipeline.

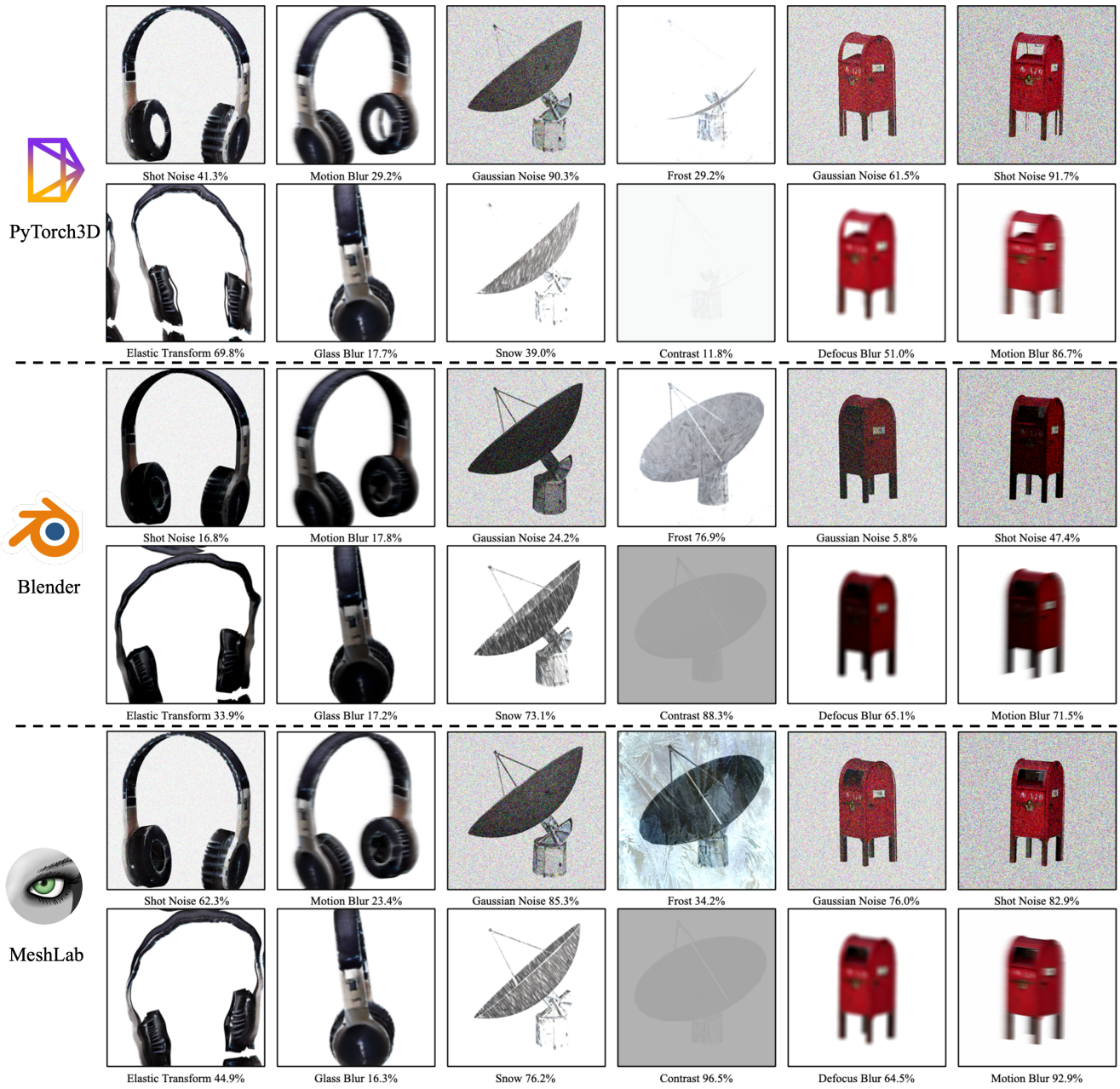

Figure 9. Performance of appearances generated by the RoNA method under different renderers. From top to bottom, the rows correspond to results obtained using PyTorch3D (the renderer used during training), Blender, and MeshLab, respectively.

In addition, we are interested in the transferability of the generated appearances. Specifically, we transfer appearances produced using ResNet-18 as the proxy model to other network architectures and observe the resulting changes in predictions. Detailed results are shown in Fig. 10. This demonstrates that the appearances generated by our RoNA method are also effective when transferred to classifiers with different architectures, rather than simply overfitting to the proxy classifier used during training. In the future, how to further improve the transferability of such robust appearances will be an important direction for further research.

9. Failure Cases Study

Although our method outperforms existing approaches in both corruption robustness and image quality, we observe that it can still exhibit instability in certain cases. For exam-

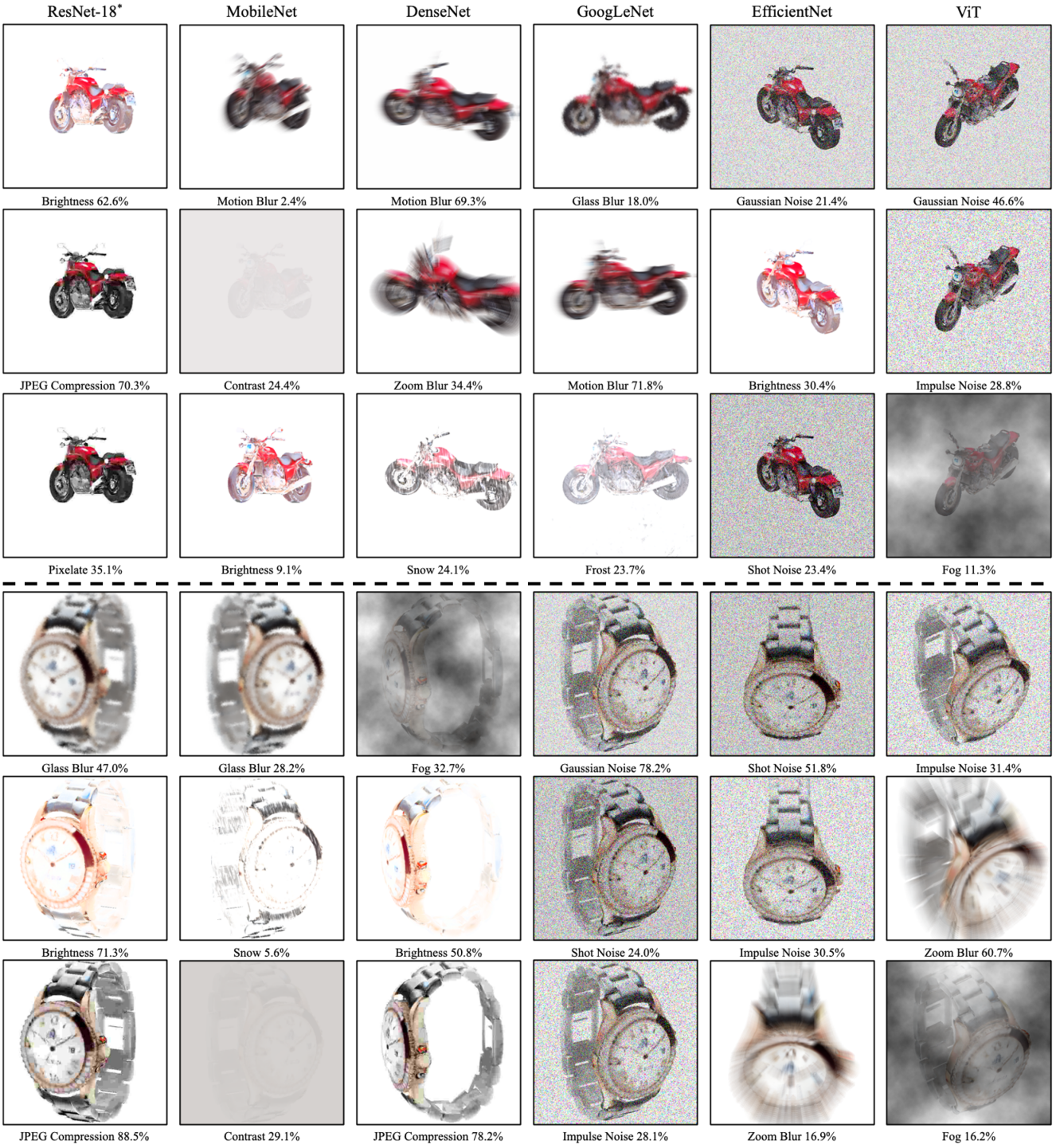

Figure 10. Performance of appearances generated by the RoNA method when transferred to different classifiers. '*' denotes the proxy classifier used during training. The erosion strength is set to 5 for all cases.

ple, our framework requires a text prompt. In general, the prompt is not crucial, so we often use a simple template like "A photo of a" + "class name". However, the prompt can be ambiguous. Fig. 11 shows one such case: when the input mesh is a computer mouse, using the prompt "A photo of a mouse" leads Stable Diffusion to interpret mouse as the an-

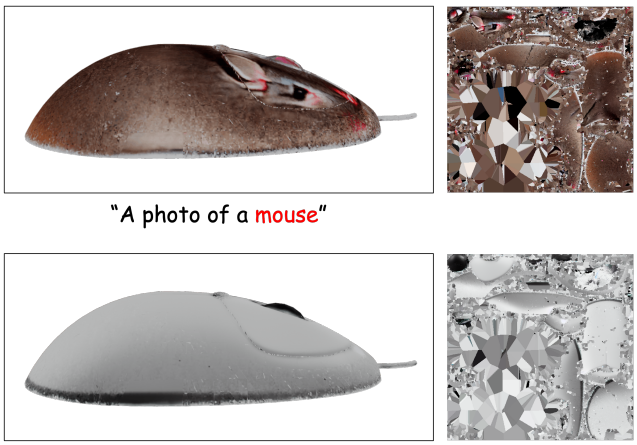

Figure 11. Failure case.

imal, producing textures with rodent-like features. A more precise description, such as "A photo of a computer mouse", is therefore required to generate textures corresponding to the computer mouse category.