Puzzle Similarity: A Perceptually-guided No-Reference Metric for Artifact Detection in 3D Scene Reconstructions

Supplementary Material

Summary In this supplementary material, we include extra details on the implementation of *PuzzleSim*, with exemplary code in Pytorch, as well as extra details on our choice of model backbone for the metric and extended qualitative results with comparisons with all tested metrics. Additionally, we include extra information on our recursive automatic inpainting application.

1. Puzzle Similarity Implementation Details

Recall from Section 3 that we can compute the quality map based on an outer product.

$$\begin{aligned} \hat{\mathcal{F}}_\ell(\mathcal{I}^{1:N}) &\in \mathbb{R}^{N\times H_\ell\times W_\ell\times C_\ell} \\ \tilde{\mathcal{F}}_\ell(\mathcal{I}^{1:N}) &= \text{flatten}\left(\hat{\mathcal{F}}_\ell(\mathcal{I}^{1:N})\right) \in \mathbb{R}^{NH_\ell W_\ell\times C_\ell} \\ S_\ell(\mathcal{I}) &= \overbrace{\text{rowmax}\underbrace{\tilde{\mathcal{F}}_\ell(\mathcal{I}_{\text{ref}}^{1:N})\otimes\tilde{\mathcal{F}}_\ell(\mathcal{I})}_{\in\mathbb{R}^{NH_\ell W_\ell\times H_\ell W_\ell}}}^{\in\mathbb{R}^{H_\ell W_\ell}} \end{aligned} \tag{1}$$

Computing this outer product naïvely would require substantial amounts of memory, as the resulting matrix before taking the maximum over rows has a dimensionality of (NHW, HW), thus NH^2W^2 elements. $N = 100$ reference images of size 128×128 would result in $26,843,545,600$ elements, requiring $\approx$ 100GB memory for 32-bit floating point numbers. We observed that this problem is similar to flash attention [1] and derived from it a memory-efficient implementation.

We leverage the fact that we are taking the maximum along rows of size (NHW). Knowing this, we can compute partial results by looping over either N, H, or W and aggregating the current maximum for each element, which reduces the memory footprint. At the same time, we take advantage of the GPU's cache hierarchy by looping in blocks. This gives an additional speedup without loss of generality. With this approach, we can cut the biggest matrix to (NbW, HW) in case we choose to aggregate along the height dimension while running over b-sized blocks with $b << H$. The final algorithm is detailed below.

```
def puzzle_similarity(F, img)
    """
    F: base model
    img: image to test
    """
    layer_similarities = []
    ref_feats = compute_normalized_features(F, refs)
    features = compute_normalized_features(F, img)
    for layer in layers:
        refs = ref_feats[layer]
        img_ = feats[layer].squeeze()
        N, C, H, W = refs.shape

        candidates = []
        # factor over h, the dimension that you max over
        block_size = 4
        for h in range(0, H, block_size):
            sim = torch.einsum(
                'cHW,nchw->nHWhw',
                img_, refs[:, :, h:h+block_size, :]
            )
            c_WH = (
                sim
                # what was rows in sim is now last dimension
                .reshape(N, H * W, -1)
                .max(dim=-1)  # distribute max over ref.W
                .values  # get max values instead of indices
                .max(dim=0)  # distribute max over ref.N
                .values  # get max values instead of indices
            )
            candidates.append(c_WH)

        sim_map = (
            torch.stack(candidates, dim=0)
            .max(dim=0)  # distribute max over ref.H
            .values
            .view(H, W)  # reshape to spatial map
        )
        sim_map = upsample(sim_map * w[layer], img.shape)
        layer_similarities.append(sim_map)

    return sum(layer_similarities)
```

In our implementation, we use a block size of 4, which reduces the matrix size to (N4W, HW), reducing memory load to only $\approx$ 3.5GB. This approach enables us to compute *PuzzleSim* efficiently even on high-resolution images, given that computing time is primarily dominated by memory fetches in our metric.

To see the impact of our blocked implementation, we compare its runtime with the naïve implementation. In Table 1, we show the results for different image sizes and number of reference images. After five warmup steps, we measured the computation time in milliseconds, averaging over 200 runs. The $\pm$ indicates half the distance between the 0.05 and 0.95 quantiles. We observe that the blocked implementation appears more stable and scales better. The experiment was performed on an NVIDIA GeForce GTX 3090 with 24GB of memory.

2. On the choice of backbone model

In Table 2 we summarize the differences we considered when choosing our backbone. While *VGG* models achieve higher performance on the ImageNet benchmark [2], model size and computational complexity are problematic in efficiency terms for our metric. Furthermore, added model capacity and improved classification performance did not seem to substantially improve the alignment of our model

Table 1. Comparison of blocked and naive implementation across different numbers of references and image sizes in ms.

Image Size	# References	PuzzleSim (blocked)	PuzzleSim (naïve)
(240, 131)	25	$3.8_{\pm 1.95}$	$2.2_{\pm 1.36}$
(240, 131)	50	$6.0_{\pm 0.58}$	$26.3_{\pm 0.13}$
(240, 131)	75	$14.7_{\pm 0.16}$	$41.4_{\pm 0.17}$
(240, 131)	100	$20.1_{\pm 0.05}$	$57.9_{\pm 0.07}$
(480, 262)	25	$5.6_{\pm 0.29}$	$4.5_{\pm 0.11}$
(480, 262)	50	$15.3_{\pm 0.08}$	$12.7_{\pm 0.09}$
(480, 262)	75	$185.7_{\pm 0.00}$	$331.8_{\pm 0.00}$
(480, 262)	100	$299.2_{\pm 0.00}$	$499.1_{\pm 0.00}$
(960, 524)	25	$56.8_{\pm 0.07}$	$57.2_{\pm 0.74}$
(960, 524)	50	$321.3_{\pm 0.00}$	$727.2_{\pm 0.00}$
(960, 524)	75	$2653.3_{\pm 0.00}$	$5421.8_{\pm 0.00}$
(960, 524)	100	$5799.1_{\pm 0.00}$	$15353.6_{\pm 0.00}$
(1920, 1048)	25	$754.4_{\pm 0.00}$	Out-of-memory
(1920, 1048)	50	$1516.7_{\pm 0.00}$	Out-of-memory
(1920, 1048)	75	$31104.0_{\pm 0.00}$	Out-of-memory
(1920, 1048)	100	$69807.4_{\pm 0.00}$	Out-of-memory

with human perception; rather hindering it. *AlexNet* and *SqueezeNet* offer much greater speed and lower memory consumption while also producing slightly preferable maps in our empirical evaluations.

Model	**#Parms**	**Acc.**	**Efficiency**	**Mem.**
VGG-19	144M	High	Low	High
VGG-16	138M	High	Low	High
AlexNet	60M	Mid	Mid	Mid
SqueezeNet	1.2M	Mid	High	Low

Table 2. Pre-trained models considered for our backbone. Accuracy refers to their relative performance on the ImageNet benchmark [2].

3. Dataset Collection Experiment Details

The experiment was run on a DELL U2718Q monitor with a consistent display setting across all participants, keeping a constant viewing distance around 70 cm under controlled lighting conditions. We recruited 23 participants (10 male, 12 female, 1 undisclosed) with a mean age of 24, all possessing normal or corrected to normal visual acuity. All test subjects were compensated for their time. We will release the dataset upon acceptance.

4. Extended Metric Validation Results

We include extensive qualitative results on all tested metrics in Figure 1 for more examples. Our metric correlates best with human assessment, even when compared with direct, full-reference metrics.

5. Automatic Recursive Inpanting Formulation Details

In this section, we provide extended details on the mathematical framework of the inpainting method. Upon sampling threshold candidates $\tau_{1:N}$ we threshold the initial quality map $\mathcal{Q}$ with each candidate to obtain the binary map M_i as shown in Eq. 7. We inpaint the current image with each candidate mask and assess their quality with our *PuzzleSim* metric:

$$\begin{aligned} \hat{\mathcal{I}}_i &= \text{Inpaint}(\mathcal{I}, M_i) \\ \hat{\mathcal{Q}}_i &= \text{PuzzleSim}(\hat{\mathcal{I}}_i) \end{aligned} \tag{2}$$

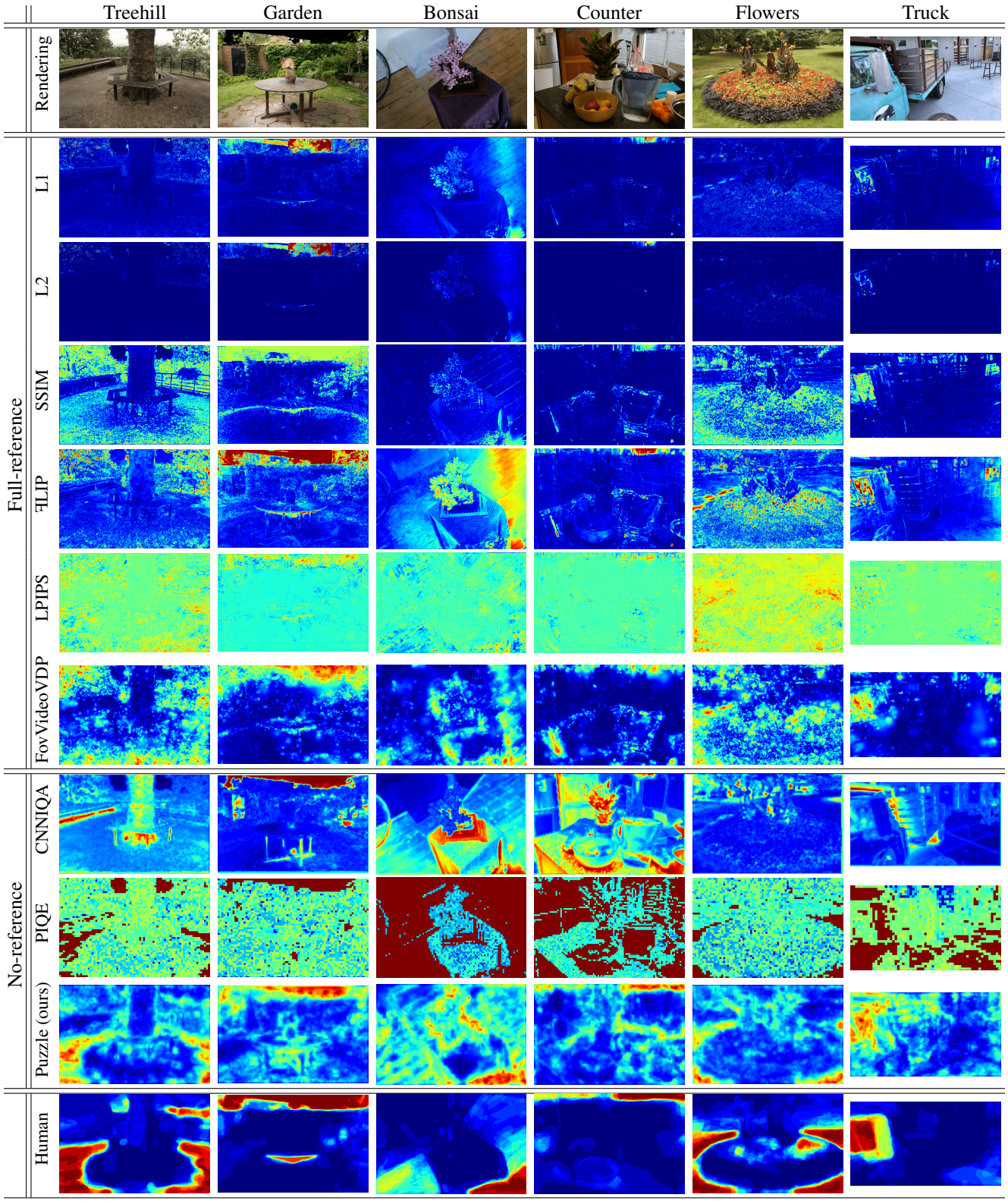

Figure 1. More elaborate comparison to all other metrics. For LPIPS we chose VGG as a backbone as it is the most popular choice.

To determine the candidate quality, we compute the average change in the quality δ_i of the inpainted regions:

$$
\begin{aligned}
|M_i| &= \sum_{h=1}^{H_{\mathcal{I}}} \sum_{w=1}^{W_{\mathcal{I}}} M_i^{(h,w)} \\
\delta_i &= \frac{1}{\lambda_i |M_i|} \underbrace{\sum_{h=1}^{H_{\mathcal{I}}} \sum_{w=1}^{W_{\mathcal{I}}} (\hat{\mathcal{Q}}_i^{(h,w)} - \mathcal{Q}^{(h,w)}) M_i^{(h,w)}}_{\text{Quality improvement of the inpainted area}} \\
\lambda_i &= |M_i|^{\frac{1}{p}}
\end{aligned}
\tag{3}
$$

where $|M_i|$ indicates the number of inpainted pixels and λ_i is a regularization term penalizing bigger masks with strength p that we empirically chose to be 4. Once the initial threshold τ_* is found, we iteratively find a new threshold in the interval $\tau_* \pm \alpha^{-1}$ std$(\hat{\mathcal{Q}}_*)$, where α is a hyperparameter that we set to 10 for all examples. By including the standard deviation, we dynamically adapt to the distribution of quality scores. As the scores become uniform, the interval becomes narrower, facilitating convergence.

References

[1] Tri Dao, View Profile, Daniel Y. Fu, View Profile, Stefano Ermon, View Profile, Atri Rudra, View Profile, Christopher Ré, and View Profile. Flashattention. *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 16344–16359, 2022. 1

[2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2015. arXiv:1409.0575 [cs]. 1, 2