The Supplementary of 'VCD-Texture: Variance Alignment based 3D-2D Co-Denoising for Text-Guided Texturing'

Shang Liu[1], Chaohui Yu[1], Chenjie Cao[1], Wen Qian[1], and Fan Wang[1]

DAMO Academy Alibaba Group, Beijing China
{liushang.ls, huakun.ych, caochenjie.ccj, qianwen.qian,
fan.w}@alibaba-inc.com

This supplementary material first presents *Additional Results*, which are organized into five sections: 1) trade-off between consistency and time, 2) time cost comparisons, 3) visual ablation studies of variance alignment, 4) visual comparisons with TexFusion, and 5) more visual comparison results. Next, the *limitations* of the proposed method are discussed. Thereafter, we offer *Algorithm Details* about the core JNP and MV-AR modules. Fourthly, we deduce the inequality of *Rasterization Variance Reduction*. Finally, further details about the *evaluation dataset* are provided.

1 Additional Results

Trade-off between Consistency, Fidelity and Time. Our experiments reveal that the number of views significantly impacts efficiency and performance. We conduct ablation studies to analyze the influence of view numbers. Tab. 1 summarizes the results, from which three key conclusions can be drawn:

- When comparing models (1) to (4), an increasing number of views results in an opposite trend between fidelity metrics (FID, ClipFID) and the consistency metric (ClipVar). More views cause more overlaps, leading to higher consistency; however, more overlap areas cause more blurriness, resulting in lower fidelity.
- Comparing models (1) to (4), the number of views and the cost time are positively correlated, more views require more generation time. Additionally, by comparing models (5) and (6), the inpainting refinement stage takes approximately 11 seconds.
- Comparing models (4) and (5), we propose a sampling view policy that utilizes 9 views during the denoising process and samples a subset of 4 views during the pixel aggregation process. This trade-off between consistency, fidelity, and time enables achieving high consistency while maintaining relatively higher fidelity.

Time Cost Comparison. We compared the runtime efficiency of VCD-Texture against prior approaches on a GPU server with eight NVIDIA RTX A800 GPUs.

Table 1: Ablations for view number on *SuxTex* dataset. LView represents latent views used in the denoising process, 4 × 90 means 4 views with a 90-degree interval each, PVnum denotes the view number used for pixel aggregation.

No.	LView	PVnum	Inpaint	FID ↓	ClipFID ↓	ClipScore ↑	ClipVar ↑	Runtime (s) ↓
(1)	4 × 90	4		**51.47**	**6.00**	31.54	82.13	**70.6**
(2)	6 × 60	6		55.32	6.77	**31.81**	83.11	93.4
(3)	8 × 45	8		57.17	7.01	31.57	83.36	98.5
(4)	9 × 40	9		59.71	7.56	31.69	**84.03**	103.4
(5)	9 × 40	4		56.05	6.88	31.62	83.94	93.1
(6)	9 × 40	4	✓	56.29	6.84	31.65	83.97	104.0

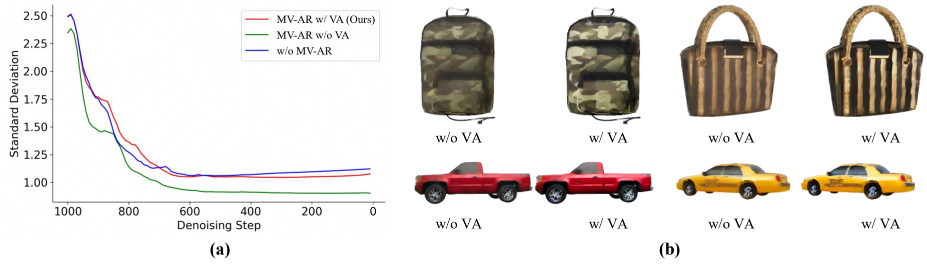
The results are presented in Tab. 2. For clarity, we organized the evaluated methods into two categories: Fitting (training-based neural networks for multi-view texture assimilation) and Re-Projection (rasterization-based re-projection without training). Re-Projection methods significantly outperformed Fitting methods, being an order of magnitude faster. Within Re-Projection, Texture [6] demonstrates the quickest performance times under default configuration, which were on par with SyncMVD [5].

Since we apply a fine mesh M_f in pixel aggregation, which takes more time in reprojecting colors to the mesh, and also introduce an additional inpainting refinement stage, this results in a marginal increase in runtime compared to Texture and SyncMVD [5]. To improve efficiency, we design a fast version of our algorithm (Ours-Fast), which incorporates three optimizations: 1) reducing denoising steps from 50 to 30; 2) remeshing the source mesh at a coarse level (from 256 to 128 resolution) to speed up color reprojection; and 3) removing the inpainting stage. This fast version achieves the fastest speed while maintaining high performance.

Table 2: Runtime Comparisons. VNum denotes the number of views, TType represents texture drawing type, and Re-Proj means texture drawing by color re-projection. 'Ours-Fast' is implemented with fewer denoising steps from 50 to 30, lower mesh resolution from 256 to 128, and without the inpainting stage, which still outperforms other competitors.

Method	TType	VNum	FID ↓	ClipFID ↓	ClipScore ↑	ClipVar ↑	Runtime (s) ↓
Text2Tex [3]	Fitting	36	112.41	16.26	30.08	81.45	842.20
Repaint3D [7]	Fitting	9	78.65	10.65	30.88	78.96	611.60
Texture [6]	Re-Proj	8	150.21	26.92	26.90	82.37	79.50
SyncMVD [5]	Re-Proj	10	65.30	16.76	28.78	81.93	83.30
Ours	Re-Proj	9	**56.29**	**6.84**	**31.65**	**83.97**	104.00
Ours-Fast	Re-Proj	9	62.57	9.73	31.42	83.10	**74.40**

Visual Ablation of Variance Alignment. The analysis of the standard deviation curve, presented in Fig. 1(a), indicates that iterative rasterization within the denoising phase results in a reduction in the magnitude of feature variance, which is likely to cause Depth-SD to produce images that are blurred or excessively smooth. To validate this conclusion, we ran experiments with and without applying variance alignment under 100 denoising steps, respectively. Fig. 1(b) shows the generated images. Comparing images without VA (first and third columns) to images with VA (second and fourth columns), we can observe that images without VA have lower contrast and much more blurriness, in contrast, images with VA exhibit higher fidelity and more distinct textural details. This proves that our proposed variance alignment can improve the image quality produced by Depth-SD when frequently conducting rasterization.



(a)

(b)

Fig. 1: The effectiveness of VA. (a) shows the standard deviation curve of three denoising policies; (b) showcases the qualitative comparison with and without VA.

Visual Comparison with TexFusion. TexFusion [1] employs a similar latent texture methodology. However, at the time of writing this manuscript, the authors had not made their code or textured mesh data publicly available. Consequently, we have opted to utilize the images presented in the original publication for visual comparison. To visualize the top view, we have incorporated two additional views (46°, 0°) and (46°, 180°) into our default horizontal viewpoints. Fig. 2 presents a visual comparison, from which we observed higher fidelity and finer details in our results.

While TexFusion [1] integrates an ancillary VGG-based loss to diminish the discrepancy between the latent and pixel domains. Nevertheless, it fails to effectively address the issue of pixel inconsistencies across views, resulting in textures that are perceptibly blurred. In contrast, our approach employs a two-stage pipeline that initially enforces consistent feature generation within the latent domain, followed by a refinement process through pixel domain inpainting. This two-stage strategy synergistically enhances the consistency and fidelity of the synthesized textures.

More Visual Comparison. As depicted in Fig. 3, we provide more qualitative comparisons against state-of-the-art counterparts, further validating the effectiveness and superiority of our proposed approach.

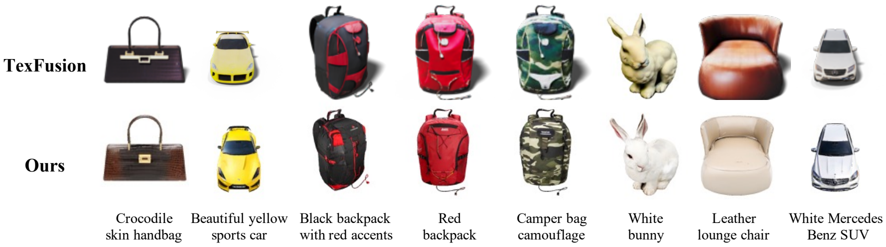


Fig. 2: Qualitative comparisons with TexFusion.

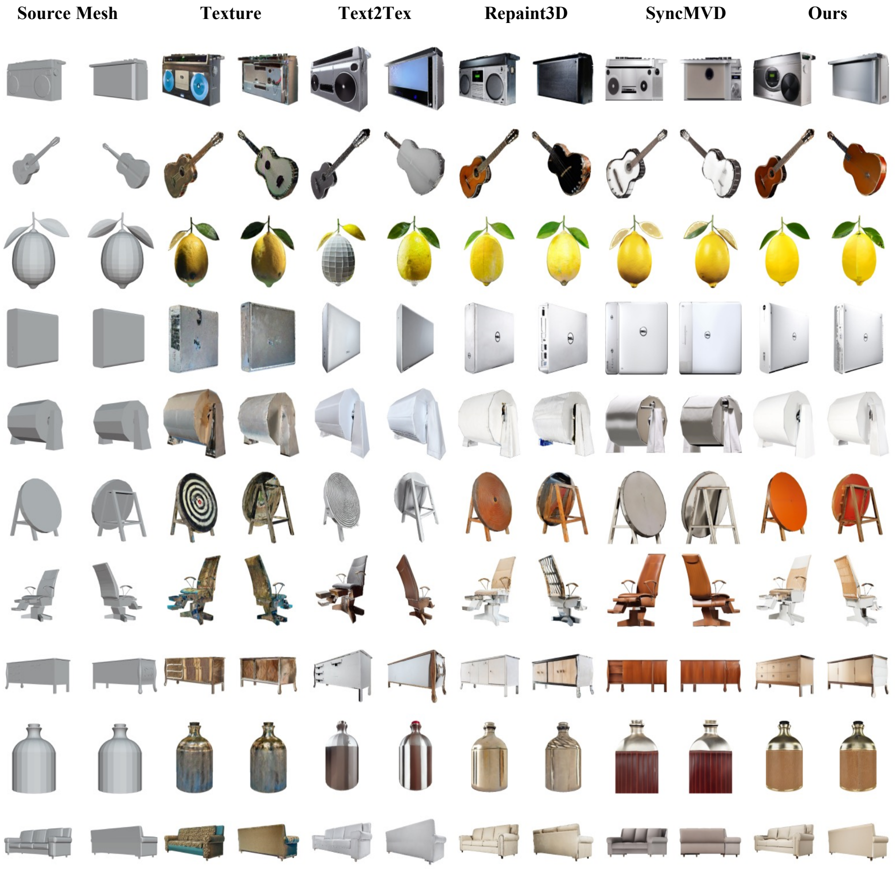


Fig. 3: Qualitative comparisons of text-guided texture synthesis. Prompts from top to down are: “CD player”, “banjo”, “lemon”, “dell inspiron white”, “Toilet paper holder with tp”, “target”, “Chair stool armchair stuhl”, “kare sideboard janus”, “flask” , “sofa”.

2 Limitations

Our research is subject to two principal limitations, primarily attributable to the constraints inherent in the pre-trained diffusion model. Firstly, the issue of pre-illumination: the images synthesized by the SD model display variations in luminance, leading to instances of local overexposure in textures. This challenge has been addressed by the development of a diffusion model [8] devoid of lighting effects. Secondly, we identify the presence of local artifacts: the Depth-SD technique synthesizes images based on a combination of depth maps and textual prompts. Due to the discrepancy in complexity between the rudimentary 3D mesh geometry and the intricate training images, the resulting depth map of the synthesized image fails to align precisely with the conditional depth map. This misalignment results in the projection of unmatched pixel colors onto the corresponding 3D vertices, thereby generating local artifacts. A potential solution to this issue lies in fine-tuning texture using methods such as PatchGAN [4], which employs a contrastive approach to learning the distribution of image patches.

3 Dataset and Evaluation details

We evaluated our method on three datasets, the statistical details of which are presented in Table 3. Notably, there are some meshes listed in TexFusion [1] were not found in the ShapeNet [2] dataset, and we utilize meshes from the same category as replacements for those invalid meshes. Since the Fréchet Inception Distance (FID) metric can be influenced by the number of evaluation images, we report the ground-truth numbers (GT-Num) in the third column of Table 3.

The comprehensive list of mesh names and prompts employed for each dataset is provided in the supplementary material, whose file names are: *subtex.txt*, *subobj.txt* and *subshape.txt*. We use the same data loader as Repaint3D [7], which can be found at: Data Loader of Repaint3D

Table 3: Evaluation Datsets.

Name	Num	GT-Num
SubTex	43	344
SubShape	445	3560
SubObj	401	3208

4 Proof of Rasterization Variance Reduction

Formally, Jensen's inequality states that if $\varphi(\cdot)$ is a convex function, and $z_{i\in N_z}$ are points in interval Z, where N_z is the number of sampling points, then for any non-negative weights λ_i that satisfy the condition: $\sum_{i=1}^{N_z} \lambda_i = 1$, the following inequality holds:

$$\varphi\left(\sum_{i=1}^{N_z} \lambda_i z_i\right) \leq \sum_{i=1}^{N_z} \lambda_i \varphi(z_i). \tag{1}$$

For random variable set $\mathbf{X_i}$, the variable $\mathbf{Y_j}$ are combined by variables sampled from $\mathbf{X_i}$, which is computed by:

$$\mathbf{Y_j} = \sum_{i=1}^{N} \lambda_i \cdot \mathbf{X_i}_j \tag{2}$$

where j denotes index in $\mathbf{Y}$, i is the variable set index. λ_i represents non-negative weights, which satisfy the condition $\sum_{i=1}^{N} \lambda_i = 1, \lambda_i \geq 0$. Let M denotes the element number of $\mathbf{Y}$, The variance of $\mathbf{Y}$ is defined by:

$$\begin{aligned} Var(\mathbf{Y}) &= \sum_{j=1}^{M} [\mathbf{Y}_j - \mu(\mathbf{Y})]^2 / (M-1) \quad (3) \\ &= \sum_{j=1}^{M} [\sum_{i=1}^{N} \lambda_i \cdot \mathbf{X_i}_j - \sum_{j=1}^{N} \lambda_i \cdot \mu(\mathbf{X_i})]^2 / (M-1) \quad (4) \\ &= \sum_{j=1}^{M} [\sum_{i=1}^{N} \lambda_i \cdot (\mathbf{X_i}_j - \mu(\mathbf{X_i})]^2 / (M-1) \quad (5) \\ & \quad (6) \end{aligned}$$

As square is a convex function, based on the Jensen's inequality 1, For each variable $\mathbf{Y}_j$, we have:

$$[\sum_{i=1}^{N} \lambda_i \cdot (\mathbf{X_i}_j - \mu(\mathbf{X_i})]^2 \leq \lambda_i \cdot \sum_{i=1}^{N} [\mathbf{X_i}_j - \mu(\mathbf{X_i})]^2 \tag{7}$$

Let $E_{Y|j}$ and $E_{X|ij}$ denote the squared deviation term in $\mathbf{Y}$ and $\mathbf{X_i}$ separately, which are defined as $E_{Y|j} = [\mathbf{Y}_j - \mu(\mathbf{Y})]^2$, $E_{\mathbf{X_i}|j} = [\mathbf{X_i}_j - \mu(\mathbf{X_i})]^2$. Referring previous inequality, we have:

$$E_{Y|j} \leq \lambda_i \cdot \sum_{i=1}^{N} E_{\mathbf{X_i}|j} \tag{8}$$

This means each squared deviation term $E_{Y|j}$ in $\mathbf{Y}$ is no large than the linear combined squared deviation term $E_{\mathbf{X_i}|j}$ in $\mathbf{X_i}$. And then apply the expectation with total number M, we have:

$$Var(\mathbf{Y}) = \sum_{j=1}^{M} E_{Y|j}/(M-1) \tag{9}$$

$$\leq \sum_{j=1}^{M} \lambda_i \cdot \sum_{i=1}^{N} E_{\mathbf{X_i}|j}/(M-1) = \sum_{i=1}^{N} \lambda_i \cdot \sum_{j=1}^{M} E_{\mathbf{X_i}|j}/(M-1) \tag{10}$$

As $Var(\mathbf{X}_i) = \sum_{j=1}^{M} E_{\mathbf{X_i}|j}/(M-1)$, thus we have:

$$Var(\mathbf{Y}) \leq \sum_{i=1}^{N} \lambda_i \cdot Var(\mathbf{X}_i) \tag{11}$$

5 Algorithm Details

We present pseudo codes for two core modules: Joint Noise Prediction in Algorithm 1, Multi-View Aggregation and Rasterization (MV-AR) in Algorithm 2. To optimize efficiency, the Joint Noise Prediction module was solely implemented at the highest resolution of the U-Net architecture. Additionally, cross-attention mechanisms operating in 3D space were also attempted but did not yield performance improvements.

References

1. Cao, T., Kreis, K., Fidler, S., Sharp, N., Yin, K.: Texfusion: Synthesizing 3d textures with text-guided image diffusion models. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4169–4181 (2023)
2. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al.: Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012 (2015)
3. Chen, D.Z., Siddiqui, Y., Lee, H.Y., Tulyakov, S., Nießner, M.: Text2tex: Text-driven texture synthesis via diffusion models. arXiv preprint arXiv:2303.11396 (2023)
4. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1125–1134 (2017)
5. Liu, Y., Xie, M., Liu, H., Wong, T.T.: Text-guided texturing by synchronized multi-view diffusion. arXiv preprint arXiv:2311.12891 (2023)
6. Richardson, E., Metzer, G., Alaluf, Y., Giryes, R., Cohen-Or, D.: Texture: Text-guided texturing of 3d shapes. arXiv preprint arXiv:2302.01721 (2023)
7. Wang, T., Kanakis, M., Schindler, K., Van Gool, L., Obukhov, A.: Breathing new life into 3d assets with generative repainting. In: Proceedings of the British Machine Vision Conference (BMVC). BMVA Press (2023)
8. Zeng, X.: Paint3d: Paint anything 3d with lighting-less texture diffusion models. arXiv preprint arXiv:2312.13913 (2023)

Algorithm 1 Joint Noise Prediction Algorithm

```
Input: Coarse mesh $M_c$, Cameras $C_n$, denoised latent feature $\mathbf{F}_n^{2D}$
Parameters: View number $N$, Vertex face index $\{f_u\}_{u=1}^{3}$ in each face, Vertex Coordinate $P_j^v$, Feature size $(w, h)$, Rasterizaiton function $R(n)$. In Pytorch3D library, $R(n)$ contain three output tensors: Depth map tensor $\hat{D}_n$, barycentric coordinate tensor $B_n$ and pixel and mesh face relation tensor $R_n^p$ specifying the indices of the faces which overlap each pixel.
Return: $\tilde{\mathbf{F}}_{l|n}^{2D}$: updated latent features at level $l$ of U-Net
procedure JOINT NOISE PREDICTION ALGORITHM
    for each level $l \in L$ do

        #Step1: Extract 3D features $\mathbf{F}_l^{3D}$
        for each view $n \in N$ do
            Build rasterization relation $R(n) = Pytorch3D.Render(M_c, C_n)$
            Compute 3D coordinates $P_{n,i}^F = \sum_{u=1}^{3} (B_{n,i}^u)^2 \cdot P_{f_u}^v$
            Extract $\mathbf{F}_l^{3D}$ from 2D foreground features with 3D coordinates.
        end for

        #Step2: Split 3D features into groups
        Compute bounding box $B^p$ of 3D space features $\mathbf{F}_l^{3D}$.
        Group $\mathbf{F}_l^{3D}$ into different groups $\mathbf{F}_{l|g}^{3D}$ by grid size $G_t$

        #Step3: Compute view-aware 2D self-attention in each 2D plane
        for each view index $n \in N$ do
            Compute $\tilde{\mathbf{F}}_{l|n}^{2D} = SelfAttn(\mathbf{F}_{l|n}^{2D})$
        end for

        #Step4: Compute grid-aware 3D self-attention in each 3D grid
        for each group index $g \in G^t$ do
            Compute $\tilde{\mathbf{F}}_{l|g}^{3D} = SelfAttn(\mathbf{F}_{l|g}^{3D})$
        end for
        Obtain 3D features $\tilde{\mathbf{F}}_{l|n}^{3D}$ in 2D space by removing coordinates of $\tilde{\mathbf{F}}_l^{3D}$

        #Step5: Combine features from 2D and 3D space
        $\tilde{\mathbf{F}}_{l|n}^{2D} = Mean(\tilde{\mathbf{F}}_{l|n}^{3D} + \tilde{\mathbf{F}}_{l|n}^{2D})$
    end for
end procedure
```

Algorithm 2 Multi-View Aggregation and Rasterization

```
Input: Coarse mesh $M_c$, Cameras $C_n$, denoised latent feature $\mathbf{F}_n^{2D}$
Parameters: View number $N$, Vertex index $\{j\}_{j=1}^{J_c}$, Feature size $(w, h)$, Upper bound of scene distance $Z_{\text{far}}$, Exponents $\tau_b, \tau_d, \tau_w$ for the power function, Rasterizaiton function $R(n)$, In Pytorch3D library, $R(n)$ contain three output tensors: Depth map tensor $\hat{D}_n$, barycentric coordinate tensor $B_n$ and pixel and mesh face relation tensor $R_n^p$ specifying the indices of the faces which overlap each pixel.
Return: $\tilde{\mathbf{X}}_n^{2D}$: updated latent predictions for each view $n \in N$
procedure MULTI-VIEW AGGREGATION AND RASTERIZATION

    #Step1: Initialize view scores $S_n$ and distance scores $D_n$ for each view $n \in N$
    for each view $n \in N$ do
        Build rasterization relation $R(n) = Pytorch3D.Render(M_c, C_n)$
        Compute view score $S_n = Cosine(Normal(M_c), Direction(C_n))$
        Compute depth score $D_n = 1 - \hat{D}_n / Z_{\text{far}}$
    end for

    #Step2:Initialize vertex features $\hat{\mathbf{X}}_{n,j}$ and vertex weights $W_{n,j}$
    for each view $n \in N$ and vertex index $j \in J_c$ do
        Compute normalization factor $\eta = \sum_i \psi(B_{n,i}, \tau_b)$
        Re-project 2D to 3D $\hat{\mathbf{X}}_{n,j}^{3D} = \sum_i X_{n,i}^{2D} \cdot \psi(B_{n,i}, \tau_b) / \eta$, the relation of each vertex $\hat{\mathbf{X}}_{n,j}^{3D}$ and 2D pixel values $\mathbf{X}_{n,i}^{2D}$ are derived from $R_n^p$.
        Compute view weight $W_{n,j} = \sum_i S_{n,i} \cdot \psi(D_{n,i}, \tau_d) \cdot \psi(B_{n,i}, \tau_b) / \eta$
    end for

    #Step3:Aggregate each view features to final texture feature
    for each vertex index $j \in J_c$ do
        Compute normalization factor $\omega = \sum_n \psi(W_{n,j}, \tau_w)$
        View Aggregation $\hat{\mathbf{X}}_j^{3D} = \sum_n \hat{\mathbf{X}}_{n,j}^{3D} \cdot \psi(W_{n,j}, \tau_w) / \omega$
    end for

    #Step4:Rasterize final texture feature to 2D plane feature
    for each view $n \in N$ do
        Compute $\tilde{\mathbf{X}}_n^{2D} = Pytorch3D.Render(M_c, C_n, \hat{\mathbf{X}}^{3D})$
        Replace background features in $\tilde{\mathbf{X}}_n^{2D}$ with $\mathbf{X}_n^{2D}$
    end for
end procedure
```