# SyncTweedies: A General Generative Framework Based on Synchronized Diffusions

**Jaihoon Kim*** **Juil Koo*** **Kyeongmin Yeo*** **Minhyuk Sung**
KAIST
{jh27kim,63days,aaaaa,mhsung}@kaist.ac.kr

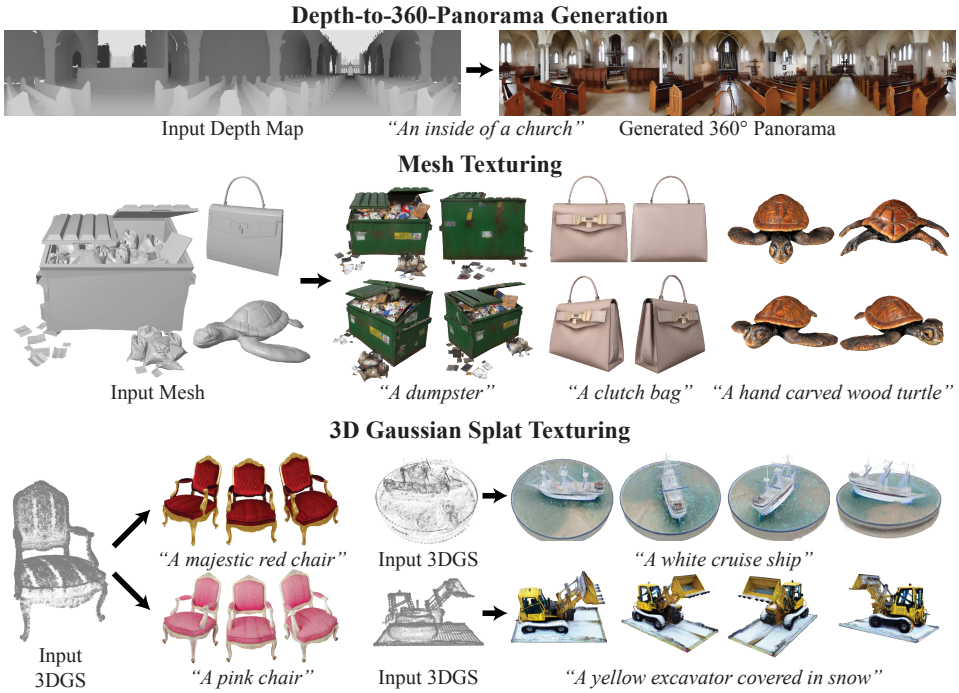



Figure 1: **Diverse visual content generated by `SyncTweedies`:** A novel synchronized diffusion process applicable to various downstream tasks without finetuning.

## Abstract

We introduce a general diffusion synchronization framework for generating diverse visual content, including ambiguous images, panorama images, 3D mesh textures, and 3D Gaussian splats textures, using a pretrained image diffusion model. We first present an analysis of various scenarios for synchronizing multiple diffusion processes through a canonical space. Based on the analysis, we introduce a novel synchronized diffusion method, `SyncTweedies`, which averages the outputs of Tweedie's formula while conducting denoising in multiple instance spaces. Compared to previous work that achieves synchronization through finetuning, `SyncTweedies` is a zero-shot method that does not require any finetuning, preserving the rich prior of diffusion models trained on Internet-scale image datasets without overfitting to specific domains. We verify that `SyncTweedies` offers the broadest applicability to diverse applications and superior performance compared to the previous state-of-the-art for each application. Our project page is at https://synctweedies.github.io.

*Equal contribution.

Preprint. Under review.

# 1 Introduction

Image diffusion models [44, 35] have shown unprecedented ability to generate plausible images that are indistinguishable from real ones. The generative power of these models stems not only from their capacity to learn from a vast diversity of potential data but also from being trained on Internet-scale image datasets [46, 47].

Our goal is to expand the capabilities of pretrained image diffusion models to produce a wide range of 2D and 3D visual content, including panoramic images and textures for 3D objects, without the need to train diffusion models for each specific visual content. Despite the existence of general image datasets on the scale of billions [46], collecting other forms of visual data at this scale is not feasible. Nonetheless, most visual content can be converted into a regular image of a specific size through certain mappings, such as projecting for panoramic images and rendering for textures of 3D objects. Thus, we employ such a *bridging* function between each type of visual content and images, along with standard image diffusion models like StableDiffusion [44] and Midjourney [35].

We introduce a general generative framework that generates data points in the desired visual content space—referred to as canonical space—by combining the denoising process of diffusion models in the conventional image space—referred to as instance spaces. Given the bridging functions connecting the canonical space and instance spaces, we first explore performing individual denoising processes in each instance space while *synchronizing* them in the canonical space via the mapping. Another approach is to denoise directly in the canonical space, although it is not immediately feasible due to the absence of diffusion models trained on the canonical space. We investigate *redirecting* the noise prediction to the instance spaces but aggregating the outputs later in the canonical space.

Depending on the timing of aggregating the outputs of computation in the instance spaces, we identify *five* main possible options for the diffusion synchronization processes. Previous works [4, 16, 32] have investigated each of the possible cases only for specific applications, and none of them have analyzed and compared them across a range of applications. For the first time, we present a general framework for diffusion synchronization processes, within which the previous works [4, 16, 32] are contextualized as specific cases. We then present extensive analyses of different choices of diffusion synchronization processes. Based on the analyses, we demonstrate that the approach which has *not* been attempted in any previous work, conducting denoising processes in *instance* spaces (not the canonical space) and synchronizing the outputs of Tweedie's formula [43] in the canonical space, provides the broadest applicability across a range of applications and the best performance. We name this approach `SyncTweedies` and showcase its superior performance in multiple visual content creation tasks compared with previous state-of-the-art methods.

Previous works [52, 31, 48, 58] finetune pretrained diffusion models to generate new types of outputs such as $360^\circ$ panoramas and 3D mesh texture images. However, this approach requires a large quantity of target content for high-quality outputs which is prohibitively expensive to acquire. When it comes to generating visual content that can be parameterized into an image, a notable zero-shot approach not utilizing diffusion synchronization is Score Distillation Sampling (SDS) [38], which has shown particular effectiveness in 3D generation and texturing [29, 55, 57, 34]. However, this alternative application of diffusion models has been observed to produce suboptimal results and also requires a high CFG [20] weight for convergence, leading to over-saturation. For 3D texture generation, specifically, an approach that iteratively updates each view image has also been explored in multiple previous works [9, 41, 7, 21, 15]. However, the accumulation of errors over iterations has been identified as a challenge. We demonstrate that our diffusion synchronization-based approach outperforms these methods in terms of generation quality across various applications.

Overall, our contributions can be summarized as follows:

- We propose, for the first time, a general generative framework for diffusion synchronization processes.
- Through extensive analyses of various options for diffusion synchronization processes, including previous works [32, 16, 60, 4], we identify that a previously *unexplored* approach, `SyncTweedies`, offers the broadest applicability and superior performance.
- In our experiments, we verify the superior performance and versatility of `SyncTweedies` across diverse applications, including texturing on 3D meshes and Gaussian Splats [24], and depth-to-360-panorama generation. Compared to the previous state-of-the-art methods based on finetuning, optimization, and iterative updates, `SyncTweedies` demonstrates significantly better results.

## 2 Problem Definition

We consider a generative process that samples data within a space we term the *canonical* space $\mathcal{Z}$, where a pretrained diffusion model is not provided. Instead, we leverage diffusion models trained in other spaces called the *instance* spaces $\{\mathcal{W}_i\}_{i=1:N}$, where a *subset* of the canonical space can be instantiated into each of them via a mapping: $f_i : \mathcal{Z} \to \mathcal{W}_i$; we refer to this mapping as the *projection*. Let $g_i$ denote the *unprojection*, which is the inverse of $f_i$, mapping the instance space to a subset of the canonical space. We assume that the entire canonical space $\mathcal{Z}$ can be expressed as a composition of multiple instance spaces $\mathcal{W}_i$, meaning that for any data point $\mathbf{z} \in \mathcal{Z}$, there exist $\{\mathbf{w}_i \,|\, \mathbf{w}_i \in \mathcal{W}_i\}_{i=1:N}$ such that

$$\mathbf{z} = \mathcal{A}\left(\{g_i(\mathbf{w}_i)\}_{i=1:N}\right), \tag{1}$$

where $\mathcal{A}$ is an aggregation function that averages the data points from the multiple instance spaces in the canonical space. Our objective is to introduce a general framework for the generative process in the canonical space by integrating multiple denoising processes from different instance spaces through synchronization.

## 3 Diffusion Synchronization

We first outline the denoising procedure of DDIM [49] and then present possible options for diffusion synchronization processes based on it.

### 3.1 Denoising Process of DDIM [49]

Song *et al.* [49] have proposed DDIM, a generalized denoising process that controls the level of randomness during denoising. In DDIM [49], the posterior of the forward process is represented as follows:

$$q_{\sigma_t}\left(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)}\right) = \mathcal{N}\left(\psi^{(t)}_{\sigma_t}(\mathbf{x}^{(t)}, \mathbf{x}^{(0)}), \sigma_t^2\mathbf{I}\right), \tag{2}$$

where $\psi^{(t)}_{\sigma_t}(\mathbf{x}^{(t)}, \mathbf{x}^{(0)}) = \sqrt{\alpha_{t-1}}\mathbf{x}^{(0)} + \sqrt{\frac{1-\alpha_{t-1}-\sigma_t^2}{1-\alpha_t}} \cdot (\mathbf{x}^{(t)} - \sqrt{\alpha_t}\mathbf{x}^{(0)})$ and $\sigma_t$ is a hyperparameter determining the level of randomness. In this paper, we consider a deterministic process where $\sigma_t = 0$ for all $t$, thus $\psi^{(t)}_{\sigma_t=0}$ will be denoted as $\psi^{(t)}$ for simplicity. During denoising process, to sample $\mathbf{x}^{(t-1)}$ from its unknown original clean data point $\mathbf{x}^{(0)}$, we estimate $\mathbf{x}^{(0)}$ using Tweedie's formula [43]:

$$\mathbf{x}^{(0)} \simeq \phi^{(t)}(\mathbf{x}^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{x}^{(t)})) = \frac{\mathbf{x}^{(t)} - \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}^{(t)})}{\sqrt{\alpha_t}}, \tag{3}$$

where $\boldsymbol{\epsilon}_\theta$ is a noise prediction network, and for simplicity, the time input and condition term in $\boldsymbol{\epsilon}_\theta$ are dropped. In short, each deterministic denoising step of DDIM [49] is expressed as follows:

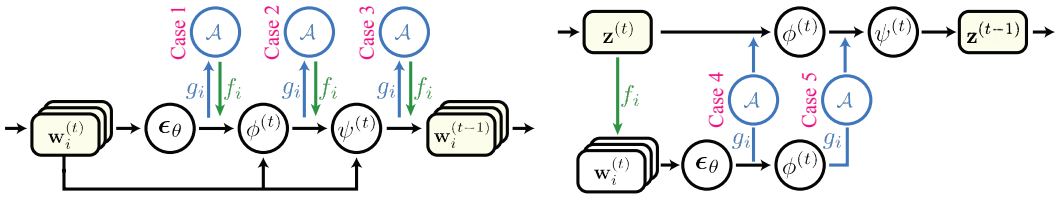
$$\mathbf{x}^{(t-1)} = \psi^{(t)}(\mathbf{x}^{(t)}, \phi^{(t)}(\mathbf{x}^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{x}^{(t)}))). \tag{4}$$

### 3.2 Diffusion Synchronization Processes

We now explore various scenarios of sampling $\mathbf{z} \in \mathcal{Z}$ by leveraging the composition of multiple denoising processes in the instance spaces $\{\mathcal{W}_i\}_{i=1:N}$. Consider the denoising step of the diffusion model at each time step $t$ in each instance space $\mathcal{W}_i$:

$$\mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))). \tag{5}$$

A naïve approach to generating data in the canonical space through the denoising processes in instance spaces would be to perform the processes independently in each instance space and then aggregate the final denoised outputs in the canonical space at the end using the averaging function $\mathcal{A}$. However, this approach results in poor outcomes that lack consistency across outputs in different instance spaces. Hence, we propose to *synchronize* the denoising processes at each time step $t$ through the unprojection operation $g_i$ from each instance space to the canonical space and the aggregation operation $\mathcal{A}$, after which the results will be back-projected via the projection operation $f_i$ to each instance space again. Note that, as described in Equation 4, the estimated mean $\psi^{(t)}(\cdot,\cdot)$ of the posterior distribution involves multiple layers of computations: noise prediction $\boldsymbol{\epsilon}_\theta(\cdot)$, Tweedie's formula [43] $\phi^{(t)}(\cdot,\cdot)$ approximating the final output $x^{(0)}$ each time step, and the final linear combination $\psi^{(t)}(\cdot,\cdot)$. Synchronization through the sequence of unprojection $g_i$, aggregation in the canonical space $\mathcal{A}$, and projection $f_i$ can thus be performed after each layer of these computations, resulting in the following three cases:




(a) Instance variable denoising process (b) Canonical variable denoising process

Figure 2: **Diagrams of diffusion synchronization processes.** The left diagram depicts denoising instance variables $\{\mathbf{w}_i\}$, while the right diagram illustrates directly denoising a canonical variable $\mathbf{z}$.

$$\text{Case 1} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, f_i(\mathcal{A}(\{g_j(\boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)}))\}_{j=1}^N))))$$

$$\text{Case 2} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, f_i(\mathcal{A}(\{g_j(\phi^{(t)}(\mathbf{w}_j^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)})))\}_{j=1}^N))$$

$$\text{Case 3} : \mathbf{w}_i^{(t-1)} = f_i(\mathcal{A}(\{g_j(\psi^{(t)}(\mathbf{w}_j^{(t)}, \phi^{(t)}(\mathbf{w}_j^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)}))))\}_{j=1}^N)).$$

In each case, we highlight the computation layer to be synchronized in red.

Another notable approach is to conduct the denoising process directly on the canonical space:

$$\mathbf{z}^{(t-1)} = \psi^{(t)}(\mathbf{z}^{(t)}, \phi^{(t)}(\mathbf{z}^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{z}^{(t)}))), \tag{6}$$

although it is not directly feasible because the noise prediction network in the canonical space $\boldsymbol{\epsilon}_\theta(\mathbf{z}^{(t)})$ is not available. Nevertheless, it can be achieved by *redirecting* the noise prediction to the instance spaces as follows:

(a) project the intermediate noisy data point $\mathbf{z}^{(t)}$ from the canonical space to each instance space, resulting in $f_i(\mathbf{z}^{(t)})$,

(b) apply a *subsequence* of the operations: $\boldsymbol{\epsilon}_\theta$, $\phi^{(t)}$, and $\psi^{(t)}$,

(c) unproject the outputs back to the canonical space via $g_i$ and then average them using the aggregation function $\mathcal{A}$, and

(d) perform the remaining operations in the canonical space.

Such an approach of performing the denoising process in the canonical space leads to the following two additional cases depending on the subsequence of operations at step (b):

$$\text{Case 4} : \mathbf{z}^{(t-1)} = \psi^{(t)}(\mathbf{z}^{(t)}, \phi^{(t)}(\mathbf{z}^{(t)}, \mathcal{A}(\{g_i(\boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)})))\}_{i=1}^N)))$$

$$\text{Case 5} : \mathbf{z}^{(t-1)} = \psi^{(t)}(\mathbf{z}^{(t)}, \mathcal{A}(\{g_i(\phi^{(t)}(f_i(\mathbf{z}^{(t)}), \boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)}))))\}_{i=1}^N)).$$

Note the analogy between Case 1 and Case 4, and Case 2 and Case 5 in terms of the information averaged in the canonical space with the aggregation operator $\mathcal{A}$: either the outputs of $\boldsymbol{\epsilon}_\theta(\cdot)$ or $\phi^{(t)}(\cdot, \cdot)$.

While it is also feasible to conduct the aggregation $\mathcal{A}$ multiple times with the output of different layers within a single denoising step, and to denoise data both in instance spaces and the canonical space, we empirically find that such more convoluted cases perform worse. In the **appendix**, we detail our exploration of all possible cases and present experimental analyses.

### 3.3 Connection to Previous Diffusion Synchronization Methods

Below, we first review previous works each corresponding to a specific case of the aforementioned possible diffusion synchronization processes while focusing on a specific application. Then, we discuss finetuning-based approaches and their limitations. In Section 4, we also review literature targeting the same applications but without synchronized diffusion.

#### 3.3.1 Zero-Shot-Based Methods

**Ambiguous Image Generation.** Ambiguous images are images that exhibit different appearances under certain transformations, such as a 90° rotation or flipping. They can be generated through

a diffusion synchronization process, considering both the canonical space $\mathcal{Z}$ and instance spaces $\{\mathcal{W}_i\}_{i=1:N}$ as the same space of the image, with the projection operation $f_i$ representing the transformation producing each appearance. Visual Anagrams [16] uses Case 4 which aggregates the noise predictions $\epsilon_\theta(\cdot)$ to generate ambiguous images.

**Arbitrary-Sized Image Generation.** In arbitrary-sized image generation, the canonical space $\mathcal{Z}$ is the space of the arbitrary-sized image, while the instance spaces $\{\mathcal{W}_i\}_{i=1:N}$ are overlapping patches across the arbitrary-sized image, matching the resolution of the images that the pretrained image diffusion model can generate. The projection operation $f_i$ corresponds to the cropping operation applied to each patch. MultiDiffusion [4] and SyncDiffusion [27] introduced arbitrary-sized image generation methods using Case 3, averaging the mean of the posterior distribution $\psi^{(t)}(\cdot,\cdot)$.

**Mesh Texturing.** In 3D mesh texturing, the texture image space serves as the canonical space $\mathcal{Z}$, and the rendered images from each view serve as the instance spaces $\{\mathcal{W}_i\}_{i=1:N}$. The rendering from the 3D textured mesh to the 2D image acts as the projection operation $f_i$. SyncMVD [32] proposed leveraging diffusion synchronization across the views using Case 5, which averages the outputs of Tweedie's formula [43] $\phi^{(t)}(\cdot,\cdot)$.

### 3.3.2 Finetuning-Based Methods

In addition to the aforementioned works, there have been attempts to achieve synchronization through finetuning. In multi-view image generation, SyncDreamer [31] and MVDream [48] finetune pretrained image diffusion models to achieve consistency across different views. For 360° panorama image generation, MVDiffusion [52] and DiffCollage [60] generate panoramas through finetuning. Additionally, Paint3D [58] trains a position encoder to directly generate 3D mesh texture images in the UV space. However, these finetuning-based methods use target sample datasets [14, 8, 13, 11] that are smaller by *orders of magnitude* compared to Internet-scale image datasets [46], e.g., 10K panorama images [8] vs. 5B images [46]. As a result, they are prone to ovefitting and losing the rich prior and generalizability of pretrained image diffusion models [44, 45]. Additionally, for 3D content, the poor quality of textures in most 3D models results in unsatisfactory texturing outcomes, even with relatively large-scale datasets [14, 13]. In our experiments, we demonstrate that our zero-shot synchronization method, fully leveraging the pretrained model without bias toward a specific dataset, provides the best realism and widest diversity, assessed by FID and KID, compared to the finetuning-based methods.

Table 1: **A quantitative comparison of ambiguous image generation.** KID [5] is scaled by $10^3$. For each row, we highlight the column whose value is within 95% of the best.

| Projection | Metric | Case 1 | SyncTweedies **Case 2** | Case 3 | Visual Anagrams [16] Case 4 | Case 5 |
|---|---|---|---|---|---|---|
| 1-to-1 Projection | CLIP-A [16] ↑ | 30.35 | 30.4 | 30.32 | 30.35 | 30.34 |
| | CLIP-C [16] ↑ | 64.52 | 64.48 | 64.49 | 64.59 | 64.48 |
| | FID [19] ↓ | 85.88 | 86.74 | 85.69 | 86.35 | 86.54 |
| | KID [5]↓ | 32.37 | 32.59 | 32.57 | 32.41 | 32.86 |
| 1-to-$n$ Projection | CLIP-A [16] ↑ | 25.97 | 30.16 | 29.94 | 25.64 | 30.23 |
| | CLIP-C [16] ↑ | 54.77 | 60.86 | 60.64 | 54.15 | 61.01 |
| | FID [19] ↓ | 232.65 | 110.51 | 117.84 | 257.53 | 108.22 |
| | KID [5] ↓ | 216.71 | 77.16 | 85.52 | 257.43 | 74.48 |
| $n$-to-1 Projection | CLIP-A [16] ↑ | 21.28 | 29.56 | 21.58 | 21.33 | 21.09 |
| | CLIP-C [16] ↑ | 49.94 | 63.1 | 50.58 | 50.05 | 50.04 |
| | FID [19] ↓ | 405.82 | 96.3 | 243.23 | 301.2 | 289.82 |
| | KID [5] ↓ | 496.98 | 40.91 | 151.11 | 233.11 | 213.45 |

## 3.4 Comparison Across the Diffusion Synchronization Processes

Here, we compare the five cases of diffusion synchronization processes in Section 3.2 and analyze their characteristics through various toy experiments.

### 3.4.1 Toy Experiment Setup: Ambiguous Image Generation

For the toy experiment setup, we employ the task of generating ambiguous images introduced by Geng *et al.* [16] (see Section 3.3.1 for descriptions of ambiguous images). We use the 95 prompt pairs and the 10 transformations used by Geng *et al.* [16], all of which are 1**-to-**1 **projections**, resulting in a total of 950 generated ambiguous images.

The quantitative and qualitative results of the five cases of diffusion synchronization processes are presented in Table 1 and Figure 3. For more detailed experiment setups, refer to the **appendix**.

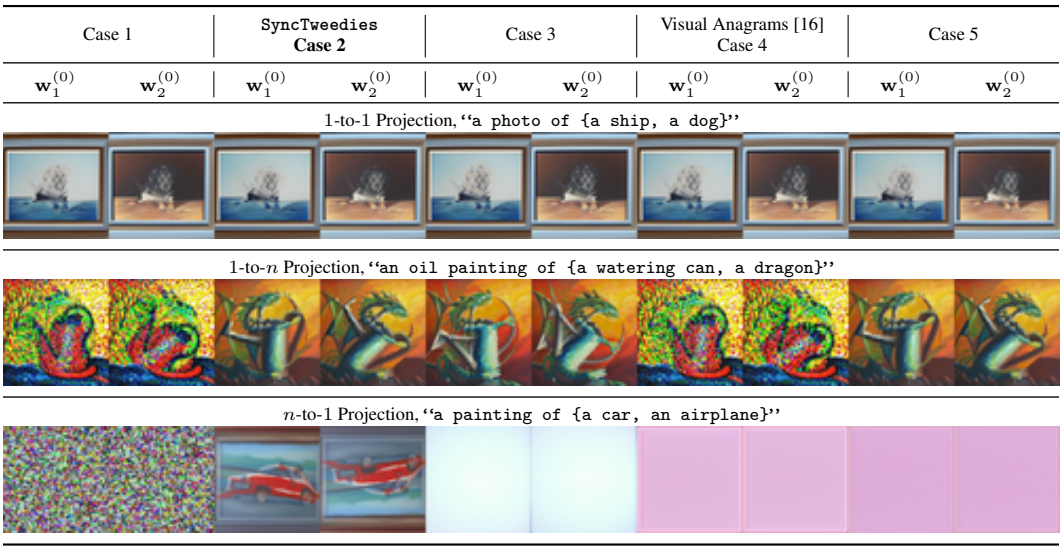



Figure 3: **A qualitative comparison of different diffusion synchronization processes.** While all cases perform well in the 1-to-1 projections, Case 1, Case 3 and Visual Anagrams [16] exhibit degraded performance when a projection is 1-to-$n$. Notably, `SyncTweedies` can be applied to the widest range of projections, including $n$-to-1 projections.

As illustrated, the results of 1-to1 projection are similar across all diffusion synchronization processes, indicating that any of those can be chosen for this specific task.

### 3.4.2 1-to-$n$ Projection

We further investigate the five cases of diffusion synchronization processes with different transformations for ambiguous images. It is important to note that all the transformations previously mentioned are perfectly invertible, meaning: $f_i(g_i(\mathbf{w}_i)) = \mathbf{w}_i$. However, in certain applications, the projection $f_i$ is often not a *function* but an 1-to-$n$ mapping, thus not allowing its inverse. For example, consider generating a texture image of a 3D object while treating the texture image space as the canonical space and the rendered image spaces as instance spaces. When mapping each pixel of a specific view image to a pixel in the texture image in the rendering process—with nearest neighbor sampling, one pixel in the texture space can be projected to multiple pixels. Hence, the unprojection operation $g_i$ cannot be a perfect inverse of the projection $f_i$ but can only be an approximation, making the reprojection error $\|\mathbf{w}_i - f_i(g_i(\mathbf{w}_i))\|$ small. We observe that such a case of having 1-to-$n$ projection $f_i$ can significantly impact the diffusion synchronization process.

As a toy experiment setup illustrating such a case with ambiguous image generation, we use rotations with nearest-neighbor sampling as transformations. We randomly select an angle and rotate an inner circle of the image while leaving the rest of the region unchanged. Due to discretization, rotating an image followed by an inverse rotation may not perfectly restore the original image.

The middle row of Table 1 and Figure 3 present the quantitative and qualitative results of this experiment. Note that the performance of Case 1 and Visual Anagrams [16], which aggregate the predicted noises $\epsilon_\theta(\cdot)$ from either instance variables $\mathbf{w}_i^{(t)}$ or a projected canonical variable $f_i(\mathbf{z}^{(t)})$ respectively, significantly declines. Also, the performance of Case 3, which aggregates the posterior means $\psi^{(t)}(\cdot,\cdot)$, shows a minor decline. Case 2 and Case 5, however, remain almost unchanged. This highlights that the denoising process is highly sensitive to the predicted noise and to the intermediate noisy data points, while it is much more robust to the outputs of Tweedie's formula [43] $\phi^{(t)}(\cdot,\cdot)$, the prediction of the final clean data point at an intermediate stage.

### 3.4.3 $n$-to-1 Projection

Then, do the results above conclude that both Case 2 and Case 5 are suitable for all applications? Lastly, we consider the case when the projection $f_i$ also involves an $n$-to-1 mapping. Such a scenario can arise when coloring not a solid mesh but a neural 3D representation rendered with the volume rendering equation [23, 24, 36]. Due to the nature of volume rendering, which involves sampling *multiple* points along a ray and taking a weighted sum of their information, the projection operation

$f_i$ includes an $n$-to-1 mapping. In this case, Case 5 results in poor outcomes due to a *variance decrease* issue. Let $\{\mathbf{x}_i\}_{i=1:N}$ be random variables, each sampled from $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \sigma_t^2\mathbf{I})$, and $\mathbf{x} = \sum_{i=1}^N w_i\mathbf{x}_i$ be the weighted sum, where $0 \leq w_i \leq 1$ and $\sum_{i=1}^N w_i = 1$. Then, $\mathbf{x}$ also follows the Gaussian distribution:

$$\mathbf{x} \sim \mathcal{N}\left(\sum_{i=1}^N w_i\boldsymbol{\mu}_i, \sum_{i=1}^N w_i^2\sigma_t^2\mathbf{I}\right). \tag{7}$$

From the triangle inequality [37], the sum of squares is always less than or equal to the square of the sum: $\sum_{i=1}^N w_i^2 \leq (\sum_{i=1}^N w_i)^2 = 1$, implying that the variance of $\mathbf{x}$ is mostly less than the variance of $\mathbf{x}_i$. Consequently, when $f_i$ includes an $n$-to-1 mapping, the variance of $\mathbf{w}_i^{(t)}$, computed as a weighted sum over multiple points in the canonical space, is less than the variance of $\mathbf{z}^{(t)}$. Thus, the final output of Case 5 becomes blurry and coarse since each intermediate noisy latent in instance spaces $\mathbf{w}_i^{(t)}$ experiences a decrease in variance compared to that of $\mathbf{z}^{(t)}$.

We validate our analysis with another toy experiment, where we use the same set of transformations used by Geng *et al.* [16] but with a multiplane image (MPI) [53] as the canonical space. The image of each instance space is rendered by first averaging colors in the multiplane of the canonical space and then applying the transformation. Ten planes are used for the multiplane image representation in our experiments. The results are presented in the third row of Table 1 and Figure 3. Notably, Case 5 also produces blurry images like the other cases, whereas Case 2 still generates realistic images.

Table 2 below summarizes suitable cases for each projection type. Note that Case 2, which has *not* been attempted in any of the previous works, is the only case that is applicable to any type of projection function. Since Case 2 involves averaging the outputs of Tweedie's formula in the instance spaces, we name this case `SyncTweedies`. Experimental results with additional applications are demonstrated in Section 5, and analysis of all possible cases is presented in the **appendix**.

Table 2: **Analysis of diffusion synchronization processes on different projection scenarios.** `SyncTweedies` offers the broadest range of applications.

| Projection | Application | Case 1 | `SyncTweedies` **Case 2** | Case 3 | Case 4 | Case 5 |
|---|---|---|---|---|---|---|
| 1-to-1 | Ambiguous images, Arbitrary-sized images | ✔ | ✔ | ✔ | ✔ | ✔ |
| 1-to-$n$ | 360° panoramas, 3D mesh texturing | ✗ | ✔ | ✗ | ✗ | ✔ |
| $n$-to-1 | 3D Gaussian Splat [24] texturing | ✗ | ✔ | ✗ | ✗ | ✗ |
| Previous Work | | - | - | MultiDiffusion [4] | Visual Anagrams [16] | SyncMVD [32] |

## 4 Related Work

In addition to Section 3.3.1 introducing previous works on diffusion synchronization, in this section, we review other previous works that utilize pretrained image diffusion models in different ways to generate or edit visual content.

**Optimization-Based Methods.** Poole *et al.* [38] first introduced Score Distillation Sampling (SDS), which facilitates data sampling in a canonical space by leveraging the loss function of the diffusion model training and performing gradient descent. This idea, originally introduced for 3D generation [54, 29, 51], has been widely applied to various applications, including vector image generation [22], ambiguous image generation [6], mesh texturing [34, 10, 57], mesh deformation [56], and 4D generation [30, 3]. Subsequent works [18, 25, 26] also proposed modified loss functions not to generate data but to edit existing data while preserving their identities. This approach, exploiting diffusion models not for denoising but for gradient-descent-based updating, generally produces less realistic outcomes and is more time-consuming compared to denoising-based generation.

**Iterative View Updating Methods.** Particularly for 3D object/scene texturing and editing, there are approaches to iteratively update each view image and subsequently refine the 3D object/scene. TEXTure [41], Text2Tex [9], and TexFusion [7] are previous works that sequentially update a partial texture image from each view and unproject it onto the 3D object mesh. For texturing 3D scene

Table 3: **A quantitative comparison in a 3D mesh texturing application.** KID is scaled by $10^3$. The best in each row is highlighted by **bold**.

| Metric | Diffusion Synchronization | | | | | Finetuning -Based | Optim. -Based | Iter. View Updating | |
|---|---|---|---|---|---|---|---|---|---|
| | Case 1 | Sync-Tweedies **Case 2** | Case 3 | Case 4 | Sync-MVD [32] Case 5 | Paint3D [58] | Paint-it [57] | TEXTure [41] | Text2Tex [9] |
| FID [19] ↓ | 135.61 | **21.76** | 36.12 | 131.67 | 22.76 | 31.66 | 28.23 | 34.98 | 26.10 |
| KID [5] ↓ | 68.63 | **1.46** | 6.60 | 65.70 | 1.74 | 5.69 | 2.30 | 6.83 | 2.51 |
| CLIP-S [39] ↑ | 25.26 | **28.89** | 27.88 | 25.31 | 28.82 | 28.04 | 28.55 | 28.63 | 27.94 |

Table 4: **A quantitative comparison in a Depth-to-360-Panorama application.** KID is scaled by $10^3$. The best in each row is highlighted by **bold**.

| Metric | Case 1 | SyncTweedies **Case 2** | Case 3 | Case 4 | Case 5 | MVDiffusion [52] |
|---|---|---|---|---|---|---|
| FID [19] ↓ | 364.61 | **42.11** | 55.95 | 348.18 | 43.39 | 80.51 |
| KID [5] ↓ | 375.42 | **21.19** | 34.67 | 362.77 | 22.87 | 56.91 |
| CLIP-S [39] ↑ | 19.75 | **28.01** | 27.19 | 19.93 | 27.99 | 24.74 |

meshes, Text2Room [21] and SceneScape [15] take a similar approach and update scene textures sequentially. Instruct-NeRF2NeRF [17] proposed to edit a 3D scene by iteratively replacing each view image used in the optimization. However, sequentially updating the canonical sample leads to error accumulations, resulting in blurriness or inconsistency across different views.

# 5 Applications

We quantitatively and qualitatively compare `SyncTweedies` with the other diffusion synchronization processes, as well as the state-of-the-art methods of each application: 3D mesh texturing (Section 5.1), depth-to-360-panorama generation (Section 5.2) ,and 3D Gaussian splats [24] texturing (Section 5.3).

**Due to the page limit, please refer to the appendix for *qualitative results*.**

Additional comprehensive experiments and detailed experiment setups, including (1) results of arbitrary-sized image generation, (2) a comparison of computation times, (3) user preference evaluations, (4) 3D mesh texture editing, and (5) implementation details of each application, are also provided in the **appendix**.

Refer to Section 3.3.1 for the detailed definition of the canonical space $\mathcal{Z}$, the instance spaces $\{\mathcal{W}_i\}_{i=1:N}$, the projection operation $f_i$, and the unprojection operation $g_i$ in each application.

**Evaluation Setup.** Across all the following applications, we compute FID [19] and KID [5] to assess the fidelity of the generated images. Additionally, we measure CLIP similarity [39] (CLIP-S) to evaluate conformity to the input text prompt. We use a depth-conditioned ControlNet [59] as the pretrained image diffusion model.

## 5.1 3D Mesh Texturing

In 3D mesh texturing, projection operation $f_i$ is a rendering function which outputs perspective view images from a 3D mesh with a texture image. This operation represents a 1-to-$n$ projection due to discretization. We evaluate five diffusion synchronization cases along with Paint3D [58], a finetuning-based method, Paint-it [57], an optimization-based method, and TEXTure [41] and Text2Tex [9], which are iterative-update-based methods. We use 429 pairs of meshes and prompts used in TEXTure [41] and Text2Tex [9].

**Results.** We present quantitative results in Table 3. The results in Table 3 align with the observations shown in the 1-to-$n$ projection case discussed in Section 3.4.2. `SyncTweedies` and SyncMVD [32] outperform other baselines across all metrics, but ours demonstrates superior performance compared to SyncMVD. Notably, `SyncTweedies` outperforms Paint3D [58], a finetuning-based method, indicating that finetuning with a relatively small set of synthetic 3D objects [14] is not sufficient for realistic texture generation. See the **appendix** for qualitative results.

Table 5: **A quantitative comparison of texturing 3D Gaussian splats [24].** KID is scaled by $10^3$. The best in each row is highlighted by **bold**.

| Metric | Diffusion Synchronization | | | | | Optim.-Based | | Iter. View. Updating |
|---|---|---|---|---|---|---|---|---|
| | Case 1 | `Sync-Tweedies` Case 2 | Case 3 | Case 4 | Case 5 | SDS [38] | MVDream-SDS [48] | IN2N [17] |
| FID [19] ↓ | 211.65 | **106.47** | 120.52 | 114.53 | 116.73 | 110.29 | 141.77 | 109.65 |
| KID [5] ↓ | 85.11 | **14.62** | 19.15 | 17.11 | 18.35 | 19.71 | 38.69 | 15.73 |
| CLIP-S [39] ↑ | 24.69 | **29.55** | 29.53 | 29.30 | 29.12 | 29.33 | 28.69 | 29.25 |

### 5.2 Depth-to-360-Panorama

We generate $360^\circ$ panorama images from input $360^\circ$ depth maps obtained from the 360MonoDepth [40] dataset. Here, the projection operation $f_i$ is a perspective transformation from the $360^\circ$ panorama canvas to a perspective view image, which is an 1-to-$n$ projection due to the discretization. We compare `SyncTweedies` with previous diffusion-synchronization-based methods [4, 16, 32] and MVDiffusion [52], which was finetuned using 3D scenes in the ScanNet [11] dataset. We generate a total of 500 $360^\circ$ panorama images at $0^\circ$ elevation, and a field of view of $72^\circ$.

**Results.** We report quantitative results of the five diffusion synchronization processes discussed in Section 3.2 in Table 4. Table 4 demonstrates a trend consistent with the 1-to-$n$ projection toy experiment results shown in Section 3.4.2. Specifically, `SyncTweedies` and Case 5, which synchronize the outputs of Tweedie's formula $\phi^{(t)}(\cdot, \cdot)$, exhibit the best performance. Notably, `SyncTweedies` demonstrates slightly superior performance across all metrics. MVDiffusion [52], which was trained using indoor scenes, fails to adapt to new, unseen domains and shows inferior results. See the **appendix** for qualitative results.

### 5.3 3D Gaussian Splats Texturing

Lastly, to verify the difference between `SyncTweedies` and Case 5 both of which demonstrate applicability up to 1-to-$n$ projections as outlined in Section 3.4.2, we explore texturing 3D Gaussian Splats [24], exemplifying an $n$-to-1 projection case. In 3D Gaussian splats texturing, the projection operation $f_i$ is an $n$-to-1 case, characterized by a volumetric rendering function [23]. This function computes a weighted sum of $n$ 3D Gaussians splats in the canonical space to render a pixel in the instance space.

While recent 3D generative models [51, 50] generate plausible 3D objects represented as 3D Gaussian splats, they often lack fine details in the appearance. We validate the effectiveness of `SyncTweedies` on pretrained 3D Gaussian splats [24] from the Synthetic NeRF dataset [36]. We use 50 views for texture generation and evaluate the results from 150 unseen views. For baselines, we evaluate diffusion-synchronization-based methods, the optimization-based methods, SDS [38], MVDream-SDS [48], and the iterative-update-based method, Instruct-NeRF2NeRF (IN2N) [17].

**Results.** Table 5 presents a quantitative comparison of 3D Gaussian splats [24] texturing. `SyncTweedies`, unaffected by the variance decrease issue, outperforms Case 5, as observed in the toy experiments in Section 3.4.3. When compared to other baselines based on optimization (SDS [38] and MVDream-SDS [48]) and iterative view updating (IN2N [17]), ours outperforms across all metrics, especially by a large margin in FID [19]. See the **appendix** for qualitative results.

## 6 Conclusion

We have explored various scenarios for synchronizing multiple denoising processes and evaluated their performance across a range of applications, including ambiguous image generation, panorama generation, and texturing on 3D mesh and 3D Gaussian splats. Our investigation demonstrates that the approach named `SyncTweedies`, which averages the outputs of Tweedie's formula while conducting denoising in multiple instance spaces, offers the best performance and the widest applicability.

**Limitations and Societal Impacts.** Despite the superior performance of `SyncTweedies` across diverse applications, updating both the geometry and appearance of 3D objects using synchronized diffusion processes remains an open problem. Also, since the pretrained image diffusion model may have been trained with uncurated images, `SyncTweedies` might inadvertently produce harmful content.

## References


[1] Luma AI. Genie.

[2] Franz Aurenhammer. Voronoi Diagrams—a survey of a fundamental geometric data structure. *CSUR*, 1991.

[3] Sherwin Bahmani, Ivan Skorokhodov, Victor Rong, Gordon Wetzstein, Leonidas Guibas, Peter Wonka, Sergey Tulyakov, Jeong Joon Park, Andrea Tagliasacchi, and David B Lindell. 4D-fy: Text-to-4d generation using hybrid score distillation sampling. *arXiv preprint arXiv:2311.17984*, 2023.

[4] Omer Bar-Tal, Lior Yariv, Yaron Lipman, and Tali Dekel. Multidiffusion: Fusing diffusion paths for controlled image generation. In *ICML*, 2023.

[5] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. In *ICLR*, 2018.

[6] Ryan Burgert, Xiang Li, Abe Leite, Kanchana Ranasinghe, and Michael S Ryoo. Diffusion illusions: Hiding images in plain sight. *arXiv preprint arXiv:2312.03817*, 2023.

[7] Tianshi Cao, Karsten Kreis, Sanja Fidler, Nicholas Sharp, and Kangxue Yin. Texfusion: Synthesizing 3d textures with text-guided image diffusion models. In *CVPR*, 2023.

[8] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *International Conference on 3D Vision (3DV)*, 2017.

[9] Dave Zhenyu Chen, Yawar Siddiqui, Hsin-Ying Lee, Sergey Tulyakov, and Matthias Nießner. Text2tex: Text-driven texture synthesis via diffusion models. In *ICCV*, 2023.

[10] Rui Chen, Yongwei Chen, Ningxin Jiao, and Kui Jia. Fantasia3D: Disentangling geometry and appearance for high-quality text-to-3d content creation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22246–22256, 2023.

[11] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017.

[12] DeepFloyd. Deepfloyd if. `https://www.deepfloyd.ai/deepfloyd-if/`.

[13] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. In *NeurIPS*, 2024.

[14] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *CVPR*, 2023.

[15] Rafail Fridman, Amit Abecasis, Yoni Kasten, and Tali Dekel. Scenescape: Text-driven consistent scene generation. In *NeurIPS*, 2024.

[16] Daniel Geng, Inbum Park, and Andrew Owens. Visual anagrams: Generating multi-view optical illusions with diffusion models. *arXiv preprint arXiv:2311.17919*, 2023.

[17] Ayaan Haque, Matthew Tancik, Alexei A Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *ICCV*, 2023.

[18] Amir Hertz, Kfir Aberman, and Daniel Cohen-Or. Delta denoising score. In *ICCV*, 2023.

[19] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2018.

[20] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS*, 2021.

[21] Lukas Höllein, Ang Cao, Andrew Owens, Justin Johnson, and Matthias Nießner. Text2room: Extracting textured 3d meshes from 2d text-to-image models. In *ICCV*, 2023.

[22] Ajay Jain, Amber Xie, and Pieter Abbeel. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In *CVPR*, 2023.

[23] James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *ACM TOG*, 1984.

[24] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM TOG*, 2023.

[25] Subin Kim, Kyungmin Lee, June Suk Choi, Jongheon Jeong, Kihyuk Sohn, and Jinwoo Shin. Collaborative score distillation for consistent visual editing. In *NeurIPS*, 2024.

[26] Juil Koo, Chanho Park, and Minhyuk Sung. Posterior distillation sampling. In *CVPR*, 2024.

[27] Yuseung Lee, Kunho Kim, Hyunjin Kim, and Minhyuk Sung. Syncdiffusion: Coherent montage via synchronized joint diffusions. In *NeurIPS*, 2023.

[28] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*, 2022.

[29] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3D: High-resolution text-to-3d content creation. In *CVPR*, 2023.

[30] Huan Ling, Seung Wook Kim, Antonio Torralba, Sanja Fidler, and Karsten Kreis. Align your gaussians: Text-to-4d with dynamic 3d gaussians and composed diffusion models. *arXiv preprint arXiv:2312.13763*, 2023.

[31] Yuan Liu, Cheng Lin, Zijiao Zeng, Xiaoxiao Long, Lingjie Liu, Taku Komura, and Wenping Wang. SyncDreamer: Generating multiview-consistent images from a single-view image. In *ICLR*, 2023.

[32] Yuxin Liu, Minshan Xie, Hanyuan Liu, and Tien-Tsin Wong. Text-guided texturing by synchronized multi-view diffusion. *arXiv preprint arXiv:2311.12891*, 2023.

[33] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *ICLR*, 2021.

[34] Gal Metzer, Elad Richardson, Or Patashnik, Raja Giryes, and Daniel Cohen-Or. Latent-NeRF for shape-guided generation of 3d shapes and textures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12663–12673, 2023.

[35] Midjourney. Midjourney. `https://www.midjourney.com/`.

[36] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2021.

[37] Dragoslav S Mitrinovic, Josip Pecaric, and Arlington M Fink. *Classical and new inequalities in analysis*. Springer Science & Business Media, 2013.

[38] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3D using 2D diffusion. In *ICLR*, 2023.

[39] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[40] Manuel Rey-Area, Mingze Yuan, and Christian Richardt. 360MonoDepth: High-resolution 360deg monocular depth estimation. In *CVPR*, 2022.

[41] Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. Texture: Text-guided texturing of 3d shapes. *ACM TOG*, 2023.

[42] Daniel Ritchie. Rudimentary framework for running two-alternative forced choice (2afc) perceptual studies on mechanical turk.

[43] Herbert E Robbins. An empirical bayes approach to statistics. In *Breakthroughs in Statistics: Foundations and basic theory*. Springer, 1956.

[44] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[45] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. In *NeurIPS*, 2022.

[46] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. LAION-5B: An open large-scale dataset for training next generation image-text models. In *NeurIPS*, 2022.

[47] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of ACL*, 2018.

[48] Yichun Shi, Peng Wang, Jianglong Ye, Mai Long, Kejie Li, and Xiao Yang. MVDream: Multi-view diffusion for 3d generation. In *ICLR*, 2024.

[49] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.

[50] Jiaxiang Tang, Zhaoxi Chen, Xiaokang Chen, Tengfei Wang, Gang Zeng, and Ziwei Liu. LGM: Large multi-view gaussian model for high-resolution 3d content creation. *arXiv preprint arXiv:2402.05054*, 2024.

[51] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. In *ICLR*, 2023.

[52] Shitao Tang, Fuyang Zhang, Jiacheng Chen, Peng Wang, and Yasutaka Furukawa. Mvdiffusion: Enabling holistic multi-view image generation with correspondence-aware diffusion. In *NeurIPS*, 2023.

[53] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. In *CVPR*, 2020.

[54] Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A Yeh, and Greg Shakhnarovich. Score jacobian chaining: Lifting pretrained 2D diffusion models for 3D generation. In *CVPR*, 2023.

[55] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. In *NeurIPS*, 2024.

[56] Seungwoo Yoo, Kunho Kim, Vladimir G Kim, and Minhyuk Sung. As-Plausible-As-Possible: Plausibility-Aware Mesh Deformation Using 2D Diffusion Priors. In *CVPR*, 2024.

[57] Kim Youwang, Tae-Hyun Oh, and Gerard Pons-Moll. Paint-it: Text-to-texture synthesis via deep convolutional texture map optimization and physically-based rendering. *arXiv preprint arXiv:2312.11360*, 2023.

[58] Xianfang Zeng, Xin Chen, Zhongqi Qi, Wen Liu, Zibo Zhao, Zhibin Wang, BIN FU, Yong Liu, and Gang Yu. Paint3d: Paint anything 3d with lighting-less texture diffusion models. In *CVPR*, 2024.

[59] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *CVPR*, 2023.

[60] Qinsheng Zhang, Jiaming Song, Xun Huang, Yongxin Chen, and Ming yu Liu. Diffcollage: Parallel generation of large content with diffusion models. In *CVPR*, 2023.

# Appendix

## Table of Contents


**A1 Qualitative Results** **14**

A1.1 3D Mesh Texturing . . . . . 14

A1.2 Depth-to-360-Panorama Generation . . . . . 15

A1.3 3D Gaussian Splats Texturing . . . . . 16

**A2 Experiment Details** **16**

A2.1 Details on Ambiguous Image Generation — Section 3.4.1 . . . . . 17

A2.2 Details on 3D Mesh Texturing — Section 5.1 . . . . . 18

A2.3 Details on Depth-to-360-Panorama Generation — Section 5.2 . . . . . 18

A2.4 Details on 3D Gaussian Splats Texturing — Section 5.3 . . . . . 19

**A3 Arbitray-Sized Image Generation** **20**

**A4 3D Mesh Texture Editing** **22**

**A5 Runtime Comparison** **23**

**A6 User Study** **24**

**A7 Analysis of Diffusion Synchronization Processes** **26**

A7.1 Overview . . . . . 26

A7.2 Instance Variable Denoising Process . . . . . 27

A7.3 Canonical Variable Denoising Process . . . . . 28

A7.4 Combined Variable Denoising Process . . . . . 29

A7.5 Quantitative Results . . . . . 29

# A1 Qualitative Results

## A1.1 3D Mesh Texturing

As shown in Figure A4, SyncTweedies and SyncMVD [32] generate the most realistic output images, aligning with the results of $n$-to-1 projection scenarios discussed in Section 3.4.2. Notably, Paint3D [58], a finetuing-based method, produces inferior textures, losing fine-details, as seen in the appearance of the car in row 1 and the patterns of the ladybug in row 5. This demonstrates the challenge of acquiring a sufficient amount of high-quality texture images for satisfactory results. The optimization-based method [57] tends to produce images with high-contrast, unnatural colors, as evidenced in rows 4 and 6. Lastly, the iterative view updating methods [41, 9] show inconsistencies across views noticeable in the front bumpers of minivan in row 1 and the fragmented fingers on baseball gloves in row 2.


Figure A4: **3D mesh texturing qualitative result.** SyncTweedies and SyncMVD [32] exhibit comparable results, outperforming other baselines. Finetuning-based method [58] produces images without fine details as it was trained on a dataset with coarse texture images. The optimization-based method [57] tends to produce unrealistic and high saturation textures, while iterative-view-updating-based methods [9, 41] show view inconsistencies.

### A1.2 Depth-to-360-Panorama Generation

As shown in Figure A5, SyncTweedies and Case 5 demonstrate the best results, aligning well with the input depth maps, with SyncTweedies showing a slightly better alignment as indicated by the red arrow in Figure A5. On the other hand, MVDiffusion [52], which is finetuned with the depth maps of indoor 3D scenes from the ScanNet [11] dataset, produces suboptimal results and fails to generate realistic 360° panoramas for out-of-domain scenes. This demonstrates that MVDiffusion [52] is overfitting to the scenes encountered during finetuning, resulting in a loss of generalizability. Cases 1 and 4, which aggregate the predicted noise $\epsilon_\theta(\cdot)$, produce noisy outputs. Case 3 yields suboptimal 360° panoramas, characterized by monochromatic appearances and a lack of detail.

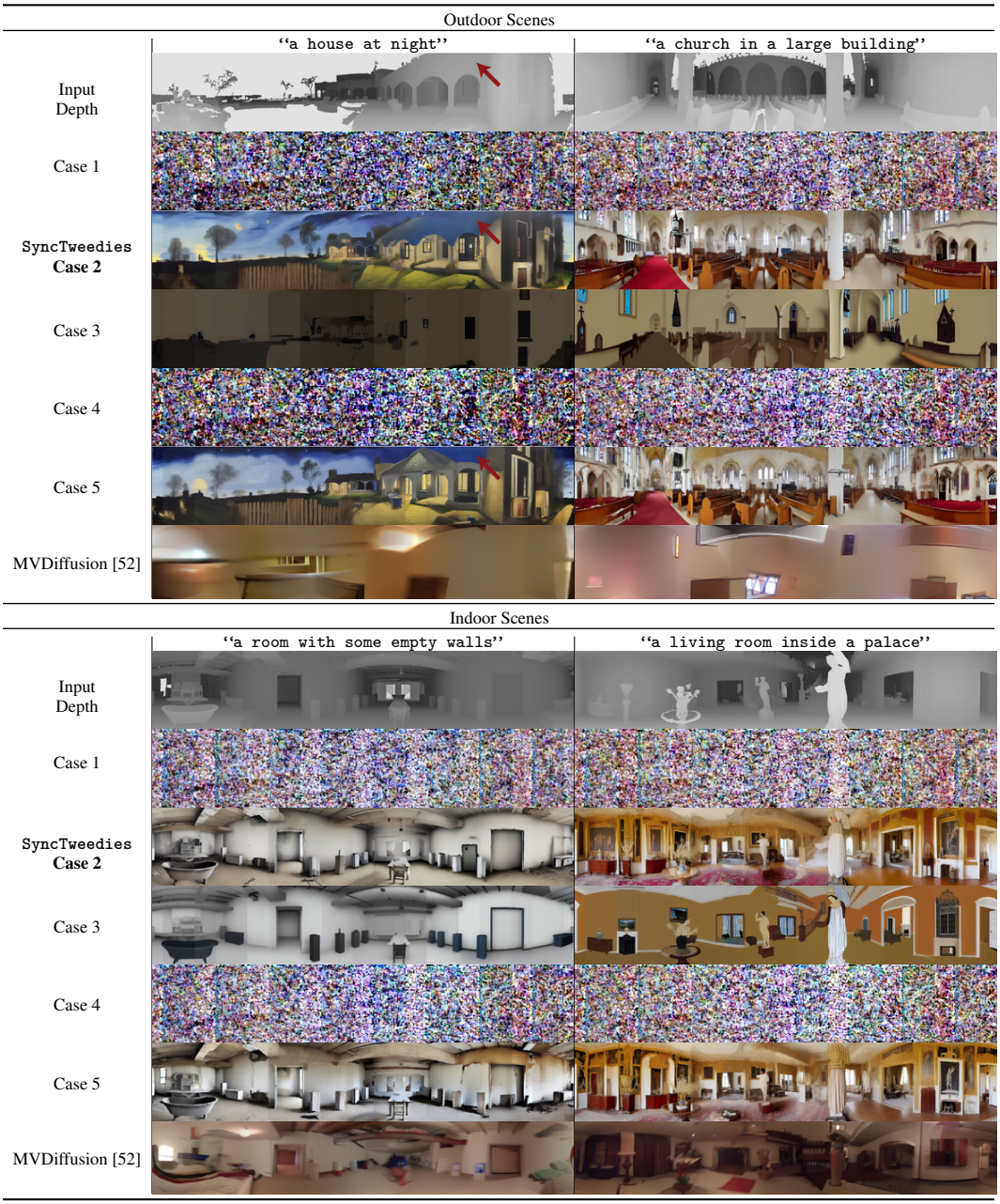



Figure A5: **Qualitative results of** 360° **panorama generation.** SyncTweedies and Case 5 generate consistent and high-fidelity 360° panorama images as observed in the 1-to-$n$ projection experiment in Section 3.4.2. MVDiffusion [52] fails to generalize to out-of-domain scenes and generates suboptimal 360° panorama images.

### A1.3 3D Gaussian Splats Texturing

Figure A6 shows that SyncTweedies generates high-fidelity results with intricate details, such as the carvings of a ship in row 1, while Case 5 lacks fine details. Optimization-based methods, SDS [38] and MVDream-SDS [48], produce artifacts characterized by high saturation, such as the corns in row 2 and the excavator in row 4. Notably, a finetuning-based method, MVDream-SDS [48], shows inferior quality to SDS. As discussed in Section 3.3.2, the poor quality of textures in the finetuning dataset [14] results in quality degradation. Iterative-view-updating-based method, IN2N [17], fails to preserve fine details, such as the head of the microphone in row 3.

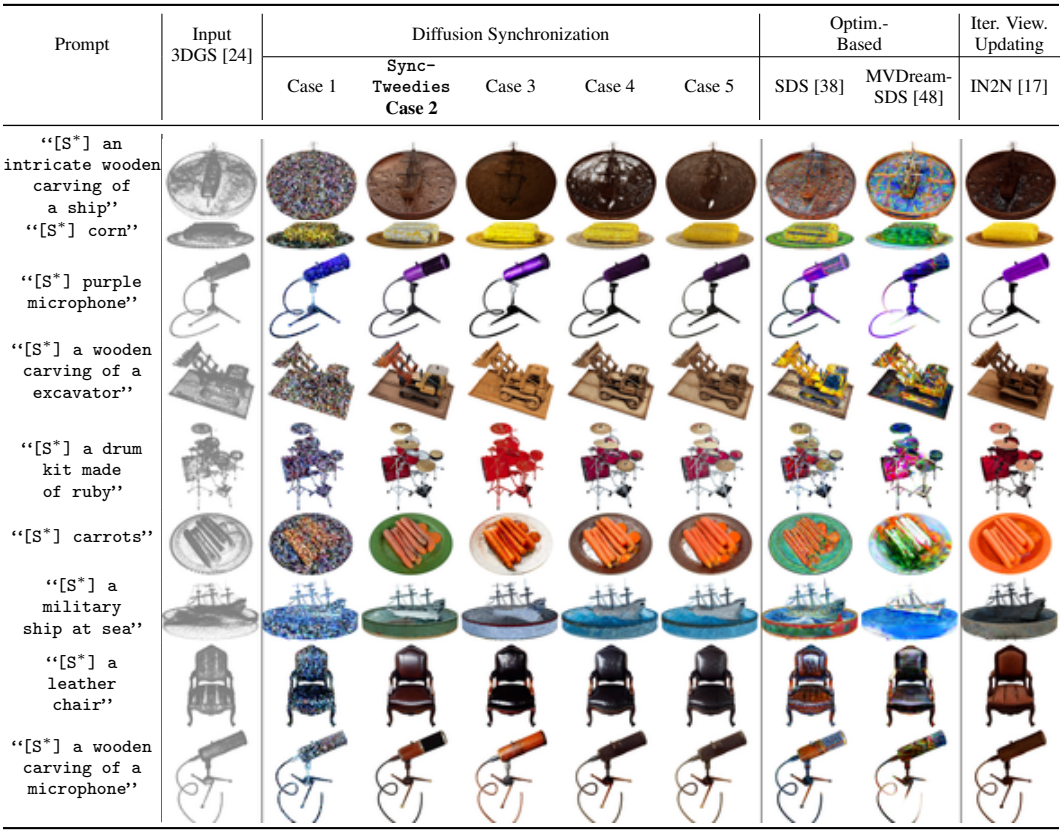



Figure A6: **Qualitative results of texturing 3D Gaussian splats.** [S*] is a prefix prompt. We used ''Make it to'' for IN2N [17] and ''A photo of'' for the other methods. Case 5 tends to lose details due to the variance decrease issue, whereas SyncTweedies generates realistic images by avoiding this issue. The optimization-based methods [38, 48] produce high contrast, unnatural colors, and the iterative view updating method [17] yields suboptimal outputs due to error accumulation.

## A2 Experiment Details

In this section, we provide details of the experiments discussed. For all diffusion synchronization processes, we use a fully deterministic DDIM [49] sampling with 30 steps, unless specified otherwise.

In the case of instance variable denoising processes introduced in Section 3.2 (Cases 1 to Case 3), we initialize instance variables by projecting an initial canonical latent $\mathbf{z}^{(T)}$ sampled from a unit Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$: $\mathbf{w}_i^{(T)} \leftarrow f_i(\mathbf{z}^{(T)})$. For $n$-to-1 projection cases, the instance variables are directly initialized from a unit Gaussian distribution which can avoid the variance decrease issue discussed in Section 3.4.1.

As described in Section 3.4.1 and Section 5, we use DeepFloyd [12] as the pretrained diffusion model for the ambiguous image generation which denoises images in the RGB space. For the depth-to-360-

panorama generation, 3D mesh texturing, and 3D Gaussian splats texturing, we employ a pretrained depth-conditioned ControlNet [59] which is based on a latent diffusion model, specifically Stable Diffusion [44]. For applications utilizing ControlNet, synchronization during the intermediate steps of diffusion synchronization processes occurs within the same latent space, except for 3D Gaussian splats texturing. In the case of 3D Gaussian splats texturing, synchronization takes place in the RGB space, and detailed explanations are provided in Section A2.4. At the end of the diffusion synchronization processes, we perform the final synchronization in the RGB space using the decoded instance variables across all applications.

**Evaluation Metrics.** For all applications, we evaluate diversity and fidelity of the generated images using FID [19] and KID [5]. These metrics compute scores based on the distance between the distribution of the generated image set and that of the reference image set, with the reference set forming the target distribution. Refer to each application section for detailed description of constructing the generated image set and the reference image set.

To evaluate the text alignment of the generated images, we report CLIP similarity score [39] (CLIP-S) which measures the similarity between the generated images $\mathbf{w}_i^{(0)}$ and their corresponding text prompts $p_i$ in CLIP [39] embedding space. Additionally, in the ambiguous image generation, we report CLIP alignment score (CLIP-A) and CLIP concealment score (CLIP-C) following previous work, Visual Anagrams [16]. To compute the metrics, we begin by calculating a CLIP similarity matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ from $N$ pairs of transformations and text prompts:

$$\mathbf{S}_{ij} = E_{\text{img}}(f_i(\mathbf{z}^{(0)}))^T E_{\text{text}}(p_j), \tag{8}$$

where $E_{\text{img}}(\cdot)$ and $E_{\text{text}}(\cdot)$ are the image encoder and the text encoder of the pretrained CLIP model [39], respectively. CLIP-A quantifies the worst alignment among the corresponding image-text pairs, specifically computed as $\min \text{diag}(\mathbf{S})$. However, this metric does not account for misalignment failure cases, where $p_i$ is visualized in $\mathbf{w}_j^{(0)}$ for $i \neq j$. CLIP-C considers alignment of an (a) image (prompt) to all prompts (images) by normalizing the similarity matrix $\mathbf{S}$ with a softmax operation:

$$\frac{1}{N}\text{tr}(\text{softmax}(\mathbf{S}/\tau)), \tag{9}$$

where $\text{tr}(\cdot)$ denotes the trace operator, and $\tau = 0.07$ is the temperature parameter of CLIP [39].

### A2.1 Details on Ambiguous Image Generation — Section 3.4.1

We present the details of the ambiguous image generation experiments in Section 3.4.1. Quantitative and qualitative results are presented in Table 1 and Figure 3.

**Evaluation Setup.** To evaluate the fidelity of the generated images using FID [19] and KID [5], we create a reference set consisting of 5,000 generated images from Stable Diffusion 1.5 [44] with the same text prompts used in the generation of ambiguous images.

**Implementation Details.** We use DeepFloyd [12] which is a two-stage cascaded pixel-space diffusion model. In the first stage, we generate $64 \times 64$ images that are upscaled to $256 \times 256$ images in the subsequent stage.

**Definition of Operations.** In the context of ambiguous image generation, both the instance variables $\{\mathbf{w}_i\}_{i=1:N}$ and canonical variables $\mathbf{z}$ share the same image space. However, instance variables exhibit different appearances from the canonical variable upon applying certain transformations.

In the 1-to-1 projection case, we use the 10 transformations used in Visual Anagrams [16], all of which are 1-to-1 mappings. The projection operation $f_i$ is defined as the transformation itself, and the unprojection operation $g_i$ is defined as the inverse of the transformation matrix.

In the scenario of 1-to-$n$ projection, we employ inner circle rotation as the projection operation $f_i$. This involves rotating the pixels within an inner circle of an image while keeping the outer pixels unchanged. The unprojection operation $g_i$ is the inverse of $f_i$. We use $14$ inner circle rotation transformations, with rotation angles evenly spaced in the range $[45°, 175°]$. For evaluation, we

utilize the same 95 prompts as in the 1-to-1 case for each transformation, generating $14 \times 95 = 1,350$ ambiguous images. After applying a rotation transformation, the grid of the rotated image does not align with the original image grid. Thus, we use the nearest-neighbor sampling to retrieve pixel colors from the original image to the rotated image. This sampling process leads to a scenario where a single pixel in the original image $\mathbf{z}$ can be mapped to multiple pixels in the rotated image $\mathbf{w}_i$, which is a 1-to-$n$ mapping.

For $n$-to-1 projection, we use the same transformations and text prompts as in the 1-to-1 projection experiment, thus resulting in a total of $10 \times 95 = 950$ ambiguous images. The only difference from the 1-to-1 projection experiment is that the canonical space variable $\mathbf{z}$ is now represented as multiplane images (MPI) [53], where a collection of planes $\{\mathbf{p}_j\}_{j=1:M}$ represents a single canonical variable. Specifically, we compute $\mathbf{z}$ by averaging the multiplane images: $\mathbf{z} = \frac{1}{M}\sum_{j=1}^{M}\mathbf{p}_j$. In the context of $n$-to-1 projection, we substitute the sequence of the unprojection $g_i$ and the aggregation $\mathcal{A}$ operation with an optimization process. The multiplane images $\mathbf{p}_j$ are optimized using the following objective function:

$$\min_{\{\mathbf{p}_j\}} \sum_{i}^{N} \left| f_i \left( \frac{1}{M} \sum_{j=1}^{M} \mathbf{p}_j \right) - \mathbf{w}_i \right|, \tag{10}$$

where wet set the number of planes $M = 10$.

### A2.2 Details on 3D Mesh Texturing — Section 5.1

We provide details of the 3D mesh texturing experiments presented in Section 5.1. Quantitative and qualitative results are shown in Table 3 and Figure A4.

**Evaluation Setup.** We use 429 mesh and prompt pairs collected from previous works, TEXTure [41] and Text2Tex [9]. For texture generation, we use eight views sampled around the object with $45^\circ$ intervals at $0^\circ$ elevation. Two additional views are sampled at $0^\circ$ and $180^\circ$ azimuths with $30^\circ$ elevation. For evaluation, we render a 3D mesh to ten perspective views with randomly sampled azimuths at $0^\circ$ elevation, resulting $10 \times 429 =$ 4,290 images. Following SyncMVD [32], the reference set images are generated by ControlNet [59] using the same depth maps and text prompts used in the texture generation.

**Implementation Details.** The resolution of the latent texture image is $1,536 \times 1,536$, and that of the latent perspective view images is $96 \times 96$. In the RGB space, the resolution of the texture image is $1,024 \times 1,024$ and that of the perspective view images is $768 \times 768$.

We adopt two approaches introduced in SyncMVD [32]: Voronoi-diagram-based filling [2] and modified self-attention layers. First, the high resolution of the latent texture image results in a texture image with sparse pixel distribution. To address this issue, we propagate the unprojected pixels to the visible regions of the texture image using the Voronoi-diagram-based filling. Second, spatially distant views tend to generate inconsistent outputs. Therefore, we adopt the modified self-attention mechanism that attends to other views when computing the attention output.

**Definition of Operations.** In the 3D mesh texturing, the canonical variable $\mathbf{z}$ is the texture image of a 3D mesh, and the instance variables $\{\mathbf{w}_i\}_{i=1:N}$ are rendered images from the 3D mesh. The projection operation $f_i$ is a rendering function where nearest-neighbor sampling is utilized to retrieve the color from the texture image to perspective view images.

As done in the $n$-to-1 projection case in Section A2.1, we replace the unprojection $g_i$ and aggregation $\mathcal{A}$ operation to an optimization process. This process optimizes the texture image $\mathbf{z}$ using the multi-view images $\{\mathbf{w}_i\}_{i=1:N}$. In the 3D mesh texturing, one pixel in the texture image $\mathbf{z}$ can be mapped to multiple pixels in a rendered image $\mathbf{w}_i$. Hence, this application corresponds to the 1-to-$n$ projection case as in Section 3.4.2.

### A2.3 Details on Depth-to-360-Panorama Generation — Section 5.2

We provide details of the Depth-to-360-Panorama generation experiments presented in Section 5.2. Refer to Table 4 and Figure A5 for quantitative and qualitative results.

**Evaluation Setup.** We evaluate `SyncTweedies` and the baseline methods on 500 pairs of $360^\circ$ panorama images and depth maps randomly sampled from 360MonoDepth [40] dataset. For each $360^\circ$ panorama image, we generate a text prompt using the output of BLIP [28] by providing a perspective view image of the panorama as input.

In the $360^\circ$ panorama generation, we use eight perspective views by evenly sampling azimuths with $45^\circ$ intervals at $0^\circ$ elevation. Each perspective view has a field of view of $72^\circ$ for synchronized-diffusion-based methods and $90^\circ$ for MVDiffusion [52]. For evaluation, we project the generated $360^\circ$ panorama image to ten perspective views with randomly sampled azimuths at $0^\circ$ elevation and a field of view of $60^\circ$. Similarly, the reference set images are obtained by projecting each ground truth $360^\circ$ panorama image into ten perspective views with azimuths randomly sampled and at $0^\circ$ elevation. In total, we use $500 \times 10 = 5,000$ perspective view images for evaluation.

**Implementation Details.** We set the resolution of a latent panorama image to 2,048 $\times$ 4,096 and that of the latent perspective view images to 64 $\times$ 64. In the RGB space, a panorama image has a resolution of 1,024 $\times$ 2,048, and perspective view images have a resolution of 512 $\times$ 512.

As done in the 3D mesh texturing, we apply the Voronoi-diagram-based filling [2] after each unprojection operation and employ the modified self-attention mechanism.

**Definition of Operations.** In the $360^\circ$ panorama generation, the canonical variable $\mathbf{z}$ represents a $360^\circ$ panorama image, while the instance variables $\{\mathbf{w}_i\}_{i=1:N}$ correspond to perspective views of the panorama. The mappings between the panorama image and the perspective views are computed as follows: First, we unproject the pixels of the perspective view image to the 3D space. Then, we apply two rotation matrices based on the azimuth and elevation angles. The pixels are then reprojected onto the surface of a unit sphere, represented as longitudes and latitudes. These spherical coordinates are finally converted to 2D coordinates on the panorama image.

Given the mappings, the projection operation $f_i$ samples colors from the panorama image using the nearest-neighbor method. Since a single pixel of a panorama image $\mathbf{z}$ can be mapped to multiple pixels of a perspective view image $\mathbf{w}_i$, the $360^\circ$ panorama generation is a 1-to-$n$ projection case, as discussed in Section 3.4.2.

### A2.4 Details on 3D Gaussian Splats Texturing — Section 5.3

We provide details of the 3D Gaussian splats texturing experiment presented in Section 5.3. Quantitative and qualitative results are provided in Table 5 and Figure A6.

**Evaluation Setup.** For evaluation, we use pretrained 3D Gaussian splats trained with multi-view images from the Synthetic NeRF dataset [36], consisting of 8 objects. We generate 40 textured 3D Gaussian splats by utilizing five different prompts per 3D object. We use 50 views for texture generation and 150 unseen views for evaluation.

**Implementation Details.** As described in Section A2, we employ ControlNet [59] which denoises latent images. To render the latent images, we replace the spherical harmonics coefficients of a 3D Gaussian splats to a 4-channel latent vector.

Additionally, we empirically observe that 3D Gaussian splats optimized in the RGB space yield better results than those optimized in the latent space. Hence, `SyncTweedies` optimizes 3D Gaussian splats in the RGB space by decoding the outputs of the Tweedie's formula. However, this approach cannot be extended to other cases that i) do not synchronize the outputs of $\phi(\cdot, \cdot)$ and ii) compute $\psi(\cdot, \cdot)$ in the canonical space. For this reason, we optimize 3D Gaussian splats in the latent space for Case 5.

**Definition of Operations.** The canonical variables $\{\mathbf{z}_j\}_{j=1:M}$ are 3D Gaussian splats and the instance space variables $\{\mathbf{w}_i\}_{i=1:N}$ are the rendered images from the 3D Gaussian splats. The projection operation $f_i$ is a volume rendering function [23, 24] where the colors (latent vectors) of multiple 3D Gaussian splats are composited to render a pixel. This corresponds to the $n$-to-1 projection as discussed in Section 3.4.3. In 3D Gaussian splats texturing, the colors of 3D Gaussian splats $\mathbf{z} = \{\mathbf{s}_j\}_{j=1:M}$ are optimized from multi-view images $\{\mathbf{w}_i\}_{i=1:N}$ as in the $n$-to-1 experiment in Section 3.4.3.

Table A6: **A quantitative comparison of arbitrary-sized image generation.** KID is scaled by $10^3$. For each row, we highlight the column whose value is within 95% of the best.

| Metric | Case 1 | `SyncTweedies` **Case 2** | MultiDiffusion [4] Case 3 | Case 4 | Case 5 |
|---|---|---|---|---|---|
| FID [19] ↓ | 32.83 | 32.82 | 32.83 | 32.82 | 32.83 |
| KID [5] ↓ | 7.79 | 7.79 | 7.79 | 7.79 | 7.80 |
| CLIP-S [39] ↑ | 31.69 | 31.69 | 31.69 | 31.69 | 31.69 |

## A3 Arbitray-Sized Image Generation

In addition to the 1-to-1 projection case presented in Section 3.4.1, we present arbitrary-sized image generation. In contrast to the $360^\circ$ panorama generation which corresponds to the 1-to-$n$ projection case, arbitrary-sized image generation involves 1-to-1 projection.

**Evaluation Setup.** We follow the evaluation setup used in SyncDiffusion [27]. Using Stable Diffusion 2.0 [44] as the pretrained diffusion model, we generate 500 arbitrary-sized images of $512 \times 3,072$ resolution per prompt. With six text prompts from SyncDiffusion [27], we generate a total of $500 \times 6 = 3,000$ arbitrary-sized images. For quantitative evaluation, we report the three metrics used in the main paper: FID [19], KID [5], and CLIP-S [39]. For the sample set, we randomly crop a partial view of each generated arbitrary-sized image to $512 \times 512$ resolution. Similarly, $3,000$ reference images with $512 \times 512$ resolution are generated from the pretrained diffusion model using the same text prompts.

**Implementation Details.** The resolution of latent arbitrary-sized image is $64 \times 384$, and the resolution of instance spaces is $64 \times 64$. We use a deterministic DDIM [49] sampling with 50 steps.

**Definition of Operations.** While both the $360^\circ$ panorama generation described in Section 5.2 and arbitrary-sized image generation involve the merging of multiple window images, arbitrary-sized image generation does not account for perspective projection. Instead, it crops a partial view of the panoramic image without considering the perspective distortion.

Note that the grid of the panoramic image $\mathbf{z}$ and the window images $\{\mathbf{w}_i\}_{i=1:N}$ are perfectly aligned. Hence, this corresponds to the 1-to-1 projection case discussed in Section 3.4.1 of the main paper.

**Result.** We report quantitative results in Table A6 and qualitative results in Figure A7. The quantitative results align with the observations shown in the 1-to-1 experiment in Section 3.4.1, where all diffusion synchronization cases show comparable performances.

This is further supported by the results in Figure A7, where all cases exhibit similar arbitrary-sized images, suggesting that any of the options can be used when the projection is 1-to-1.

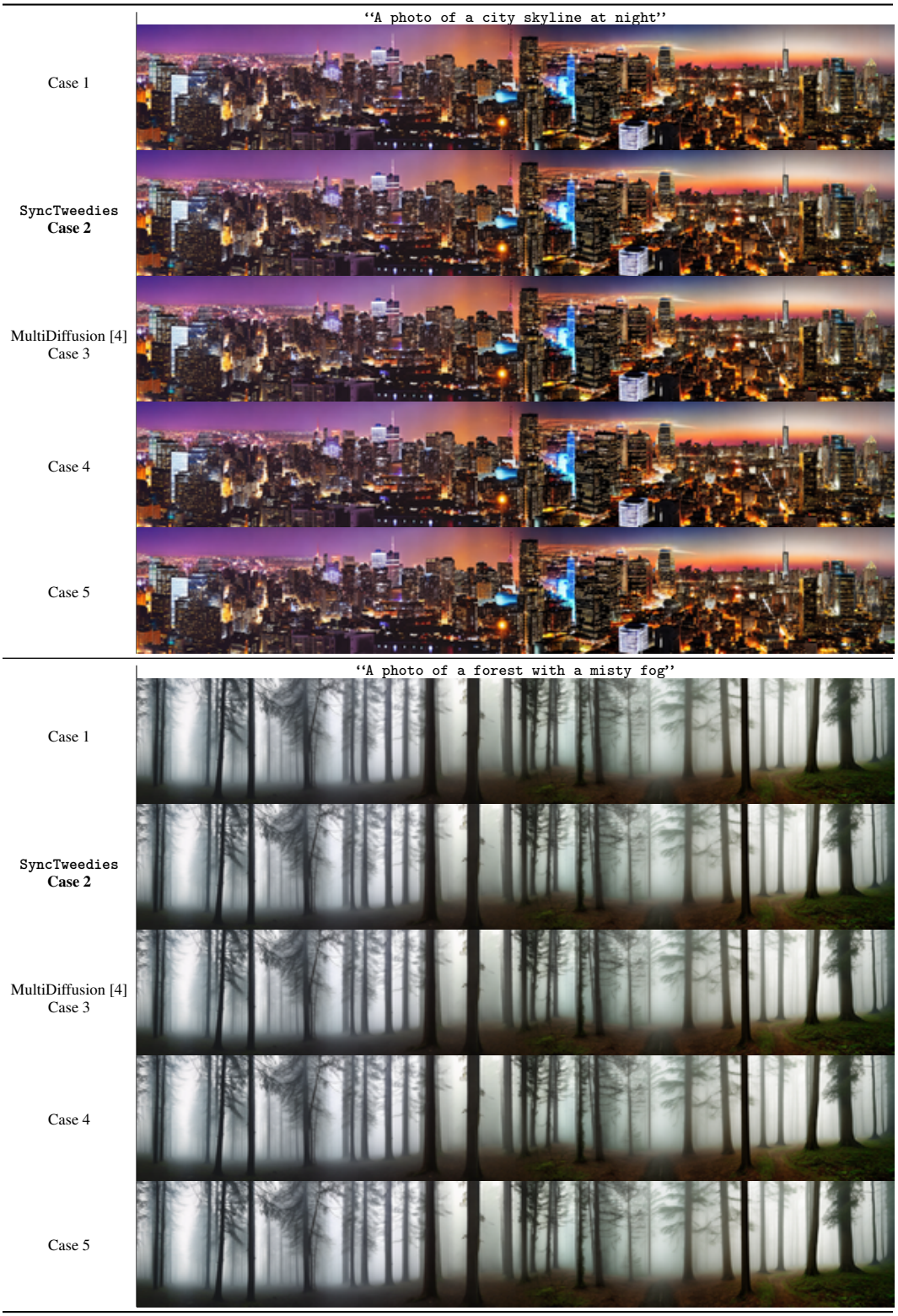



Figure A7: **Qualitative results of arbitrary-sized image generation.** All cases of diffusion synchronization processes show comparable results in the 1-to-1 projection.

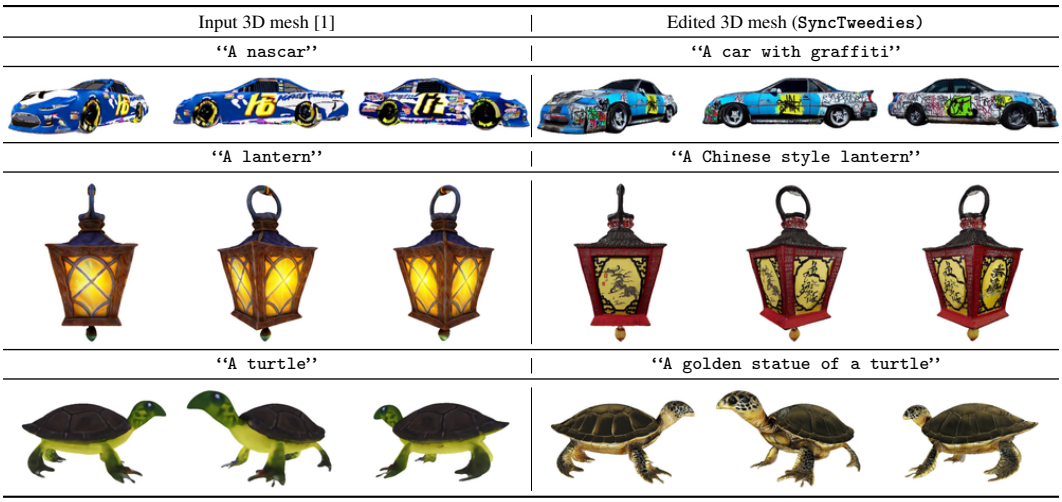



Figure A8: **Qualitative results of 3D mesh texture editing.** We edit the textures of the 3D meshes generated from *Genies* [1] using SyncTweedies.

## A4 3D Mesh Texture Editing

In this section, we extend the 3D mesh texture generation in Section 5.1, and present texture editing application.

Despite the recent successes of 3D generation models [1, 31], the textures of the generated 3D meshes often lack fine details. We utilize SyncTweedies to edit the textures of the generated 3D meshes, and enhance the texture quality. Specifically, we use the 3D meshes generated from a text-to-3D model, Genie [1].

We follow SDEdit [33] to edit the textures of the 3D mesh. We begin by adding noise at intermediate time $t'$ to the texture image of the 3D mesh, and take a reverse process starting from the same intermediate time $t'$.

**Implementation Details.** We set the CFG weight [20] to 30 and $t'$ to 0.8. For other settings, we follow the 3D mesh texture generation experiment presented in Section 5.1.

**Results.** We present qualitative results of 3D mesh texture editing in Figure A8. The 3D meshes edited with SyncTweedies exhibit fine details, including graffiti on the car in row 1, paintings on the lantern in row 2, and the intricate shells of the turtle in row 3.

Table A7: **A runtime comparison in 3D mesh texturing and 3D Gaussian splats texturing applications.** The best in each row is highlighted by **bold**.

| Metric | Diffusion Synchronization | Finetuning. -Based | Optim. -Based | Iter. View Updating | |
|---|---|---|---|---|---|
| Runtime (minutes) ↓ | 3D Mesh Texturing | | | | |
| | **SyncTweedies Case 2** | Paint3D [58] | Paint-it [57] | TEXTure [41] | Text2Tex [9] |
| | 1.83 | 2.65 | 21.95 | **1.54** | 13.10 |
| | 3D Gaussian Splats Texturing | | | | |
| | **SyncTweedies Case 2** | - | SDS [38] | IN2N [17] | |
| | **10.56** | - | 85.50 | 37.93 | |

## A5 Runtime Comparison

As discussed in Section 4, one of the advantages of diffusion synchronization processes is the fast computational speed. We compare the runtime performance of SyncTweedies with optimization-based and iterative-update-based methods in the 3D mesh texturing and the 3D Gaussian splat texturing. The quantitative results are presented in Table A7.

In the 3D mesh texturing, SyncTweedies shows faster computation times than other baselines except TEXTure [41] which shows comparable running time. However, TEXTure [41] generates suboptimal texture outputs as observed in Table 3 and Figure A4. The finetuning-based method Paint3D [58] has a comparable running time to SyncTweedies, but it shows inferior quality, as seen in Table 3 and Figure A4. Another iterative-update-based method, Text2Tex [9], improves quality of texture image by integrating an additional refinement module, it introduces additional overhead in terms of running times. In contrast, SyncTweedies achieves running times that are 7 times faster than Text2Tex and even outperforms across all metrics as shown in Table 3. Lastly, SyncTweedies shows 11 times faster running time when compared to Paint-it [57], an optimization-based method.

In the 3D Gaussian splats texturing, SyncTweedies achieves the fastest running time. SyncTweedies is 3 times faster than the iterative-update-based method IN2N [17], and 8 times faster than the optimization-based method, SDS [38]. This shows that SyncTweedies not only generates high-fidelity textures, but also excels other baselines in computational speed.

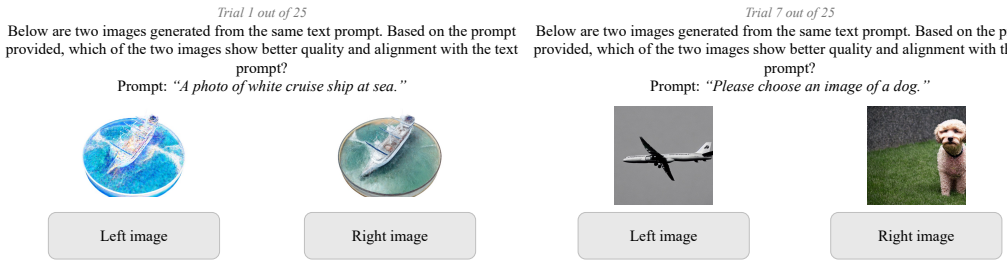
## A6 User Study

We conduct user studies to evaluate the textures of 3D Gaussian splats [24] through Amazon's Mechanical Turk. Following the methodology of Ritchie [42], participants were presented with input text prompts and randomly sampled output images generated by our method and the baseline methods. Participants are asked to choose the most plausible image that aligns with the given text prompt. In Table A8, our results are the most preferred in the human evaluations compared to the other baselines.

**Details of User Study** We conduct separate user studies comparing our method against representative baselines for diffusion synchronization methods (Case 5), optimization methods (SDS [38], MVDream-SDS [48]), and iterative view updating methods (IN2N [17]). For each user study, we use 20 images in a shuffled order including five vigilance tasks. We collected survey responses only from participants who passed the vigilance tasks. Specifically, 94 out of 100 participants passed in the study with Case 5, 90 out of 100 passed with SDS [38], 95 out of 100 passed with MVDream-SDS [48], and 92 out of 100 passed with IN2N [17]. Screenshots of our user studies, including an example of vigilance tasks, are displayed in Figure A9.

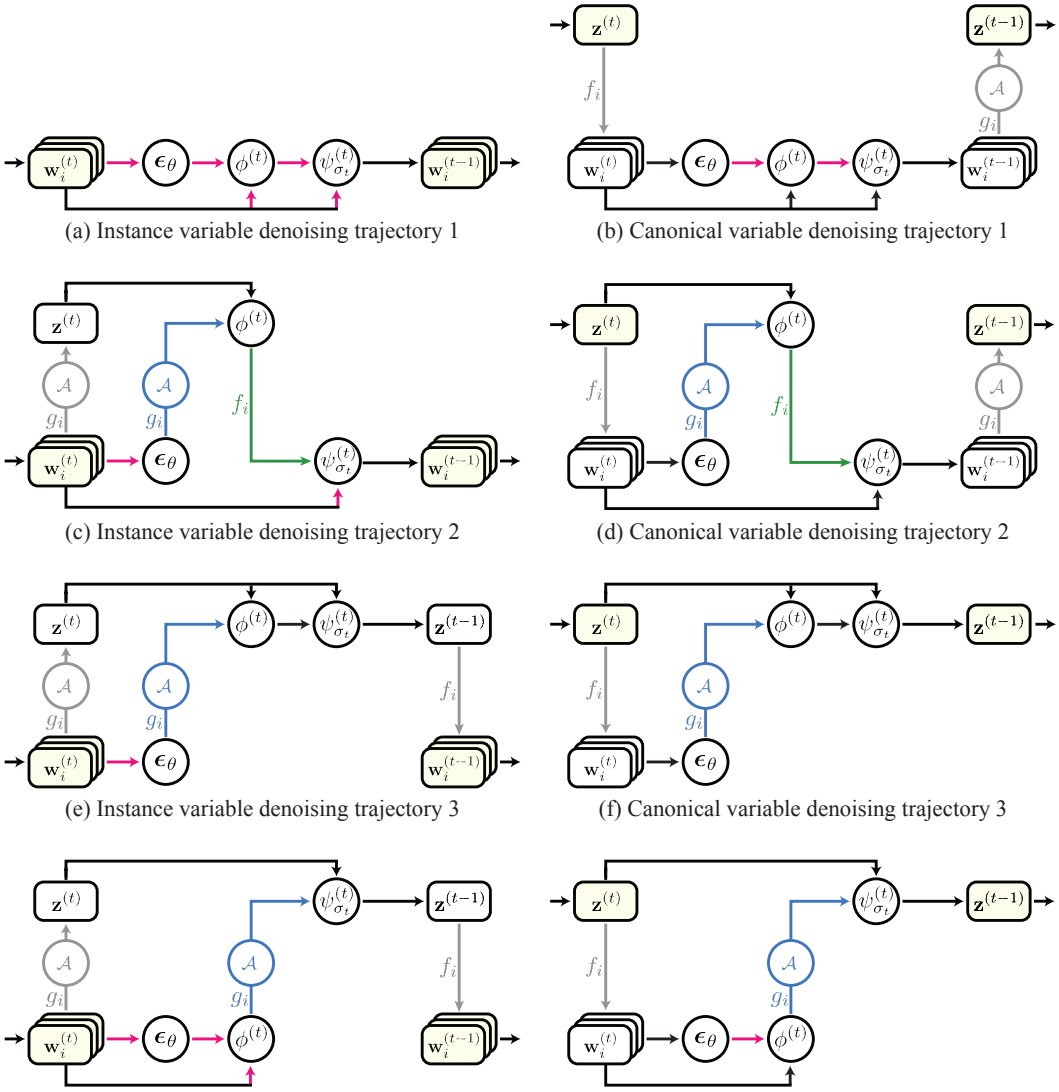
Table A8: **User study results in 3D Gaussian splats texturing application.** `SyncTweedies` is the most preferred method among the baselines from human evaluators.

| Baselines | Case 5 | SDS [38] | MVDream-SDS [48] | IN2N [17] |
|---|---|---|---|---|
| Prefer Baseline (%) | 33.56 | 41.33 | 12.21 | 40.05 |
| Prefer `SyncTweedies` (%) | **66.44** | **58.67** | **87.79** | **59.95** |




(a) Main problem (b) Vigilance task

Figure A9: **3D Gaussian splats texturing user study screenshots.** The participants are presented with generated images and an input prompt, and are asked to select an image that shows better quality and alignment with the prompt.



(a) Instance variable denoising trajectory 1
(b) Canonical variable denoising trajectory 1
(c) Instance variable denoising trajectory 2
(d) Canonical variable denoising trajectory 2
(e) Instance variable denoising trajectory 3
(f) Canonical variable denoising trajectory 3
(g) Instance variable denoising trajectory 4
(h) Canonical variable denoising trajectory 4

Figure A10: **Diagrams of diffusion synchronization processes.** All feasible trajectories of the instance variable denoising process (left) and the canonical variable denoising process (right). Each row shares the same trajectory with different variables denoised.

## A7 Analysis of Diffusion Synchronization Processes

As outlined in Section 3.4.3, we present a comprehensive analysis of all possible diffusion synchronization processes, including the representative five diffusion synchronization processes introduced in Section 3.2. Following the main paper, we categorize diffusion synchronization processes into two types: the *instance variable denoising process*, where instance variables $\{\mathbf{w}_i^{(t)}\}$ are denoised, and the *canonical variable denoising process*, which denoises a canonical variable $\mathbf{z}^{(t)}$ directly. Unlike the representative cases, other all feasible cases either take inconsistent inputs when computing $\boldsymbol{\epsilon}_\theta(\cdot)$, $\phi^{(t)}(\cdot,\cdot)$ and $\psi^{(t)}(\cdot,\cdot)$ or conduct the aggregation $\mathcal{A}$ multiple times. Additionally, for a more exhaustive analysis, we introduce another type of diffusion synchronization processes, named the *combined variable denoising process*, which denoises $\{\mathbf{w}_i^{(t)}\}$ and $\mathbf{z}^{(t)}$ together.

We present a total of 46 feasible cases for the instance variable denoising process, 8 for the canonical variable denoising process, and an additional 6 representative cases for the combined variable denoising process. We provide instance variable denoising cases in Section A7.2, and canonical variable denoising cases in Section A7.3. Additionally, the six representative cases for the combined variable denoising process are detailed in Section A7.4.

We conduct a quantitative comparison of all listed cases following the experiment setup outlined in Section 3.4.1, and the results are presented in Section A7.5.

### A7.1 Overview

We provide the representative trajectories in Figure A10, where (a)-(b), (c)-(d), (e)-(f), and (g)-(h) follow the same trajectory but differ in the denoising variable, either instance or canonical, respectively. In each denoising case, there are $2^2 = 4$ possible trajectories determined by whether $\phi^{(t)}(\cdot,\cdot)$ and $\psi^{(t)}(\cdot,\cdot)$ are computed in the canonical space or instance space. This is because among the three computation layers—$\boldsymbol{\epsilon}_\theta(\cdot)$, $\phi^{(t)}(\cdot,\cdot)$ and $\psi^{(t)}(\cdot,\cdot)$—only the last two operations can be computed in both the canonical space and the instance space unlike noise prediction which is only available in the instance space. Table A9 summarizes the computation spaces of $\phi^{(t)}(\cdot,\cdot)$ and $\psi^{(t)}(\cdot,\cdot)$, along with their corresponding trajectories.

Table A9: **Computation space of each denoising trajectory.** Except for the noise prediction $\boldsymbol{\epsilon}_\theta(\cdot)$, $\phi^{(t)}(\cdot,\cdot)$ and $\psi^{(t)}(\cdot,\cdot)$ can be computed in either instance space $\mathcal{W}_i$ or canonical space $\mathcal{Z}$.

| Trajectory | $\phi^{(t)}(\cdot,\cdot)$ Computation space | $\psi^{(t)}(\cdot,\cdot)$ Computation space |
|---|---|---|
| Trajectory 1 | $\mathcal{W}_i$ | $\mathcal{W}_i$ |
| Trajectory 2 | $\mathcal{Z}$ | $\mathcal{W}_i$ |
| Trajectory 3 | $\mathcal{Z}$ | $\mathcal{Z}$ |
| Trajectory 4 | $\mathcal{W}_i$ | $\mathcal{Z}$ |

Next, we introduce an additional operator $\mathcal{F}_i$ that synchronizes instance variables. This operator unprojects a set of instance variables and averages them in the canonical space. Subsequently, the aggregated variables are reprojected to the instance space:

$$\mathcal{F}_i(\{\mathbf{w}_j\}_{j=1:N}) = f_i(\mathcal{A}(\{g_j(\mathbf{w}_j)\}_{j=1:N})). \tag{11}$$

The red arrows in the diagrams of Figure A10 indicate the potential incorporation of $\mathcal{F}_i$. Thus, a total of $2^N$ different cases can be derived from a trajectory marked by $N$ red arrows, depending on whether $\mathcal{F}_i$ is applied to each variable or not.

Lastly, we review the five representative diffusion synchronization processes discussed in Section 3.2, along with two additional denoising processes: an instance variable denoising process that proceeds without synchronization and a canonical variable denoising process that averages the outputs of $\psi^{(t)}(\cdot,\cdot)$ (Case 6):

$$\begin{aligned}
\text{No Synchronization} &: \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))) \\
\text{Case 1} &: \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))) \\
\text{Case 2} &: \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))
\end{aligned}$$

$$\text{Case 3} : \mathbf{w}_i^{(t-1)} = \mathcal{F}_i(\psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))$$
$$\text{Case 4} : \mathbf{z}^{(t-1)} = \psi^{(t)}(\mathbf{z}^{(t)}, \phi^{(t)}(\mathbf{z}^{(t)}, \mathcal{A}(\{g_i(\boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)})))\})))$$
$$\text{Case 5} : \mathbf{z}^{(t-1)} = \psi^{(t)}(\mathbf{z}^{(t)}, \mathcal{A}(\{g_i(\phi^{(t)}(f_i(\mathbf{z}^{(t)}), \boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)}))))\}))$$
$$\text{Case 6} : \mathbf{z}^{(t-1)} = \mathcal{A}(\{g_i(\psi^{(t)}(f_i(\mathbf{z}^{(t)}), \phi^{(t)}(f_i(\mathbf{z}^{(t)}), \boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)})))))\}).$$

Note that Case 3 and 6 are identical except for the initialization, which can be either $\{\mathbf{w}_i^{(T)}\}$ or $\mathbf{z}^{(T)}$.

For the independent instance variable denoising process (No Synchronization), we apply the final synchronization in the RGB space at the end of the denoising process.

### A7.2 Instance Variable Denoising Process

Here, we explore all possible instance variable denoising processes. In these processes, the canonical space $\mathcal{Z}$ is employed to *synchronize* the outputs of $\boldsymbol{\epsilon}_\theta(\cdot)$, $\phi^{(t)}(\cdot,\cdot)$ and $\psi^{(t)}(\cdot,\cdot)$ in the instance spaces.

Following the trajectory 1 shown in part (a) of Figure A10, marked by five red arrows, there are a total of $2^5 = 32$ possible denoising processes. This includes the independent instance variable denoising process (No Synchronization), where $\mathcal{F}_i$ is not applied at any red arrow. Additionally, the three representative instance variable denoising processes, Cases 1 to 3, are also included, along with Cases 7 to 34 which are presented below:

$$\text{Case 7} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)}))))$$
$$\text{Case 8} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 9} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))$$
$$\text{Case 10} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)}))))$$
$$\text{Case 11} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))$$
$$\text{Case 12} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 13} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 14} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 15} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)})))))))$$
$$\text{Case 16} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))$$
$$\text{Case 17} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 18} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 19} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)})))))))$$
$$\text{Case 20} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))$$
$$\text{Case 21} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)}))))$$
$$\text{Case 22} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))$$
$$\text{Case 23} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 24} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))$$
$$\text{Case 25} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 26} : \mathbf{w}_i^{(t-1)} = \mathcal{F}_i(\psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 27} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))))$$
$$\text{Case 28} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)}))))))$$
$$\text{Case 29} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))))$$

$$\text{Case 30} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_i^{(t)}))))))$$

$$\text{Case 31} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))$$

$$\text{Case 32} : \mathbf{w}_i^{(t-1)} = \mathcal{F}_i(\psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))))$$

$$\text{Case 33} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))))$$

$$\text{Case 34} : \mathbf{w}_i^{(t-1)} = \mathcal{F}_i(\psi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))))).$$

Similarly, four cases are derived from the trajectory 2 shown in part (c) of Figure A10. These correspond to Cases 35 to 38 below:

$$\text{Case 35} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, f_i(\phi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)}))\}))))$$

$$\text{Case 36} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, f_i(\phi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_j^{(t)}))\}))))$$

$$\text{Case 37} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), f_i(\phi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)}))\}))))$$

$$\text{Case 38} : \mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathcal{F}_i(\mathbf{w}_i^{(t)}), f_i(\phi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_j^{(t)}))\}))))).$$

The trajectory 3 shown in part (e) of Figure A10 accounts for two cases, corresponding to Case 39 and Case 40 below:

$$\text{Case 39} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \phi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)}))\}))))$$

$$\text{Case 40} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \phi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_j^{(t)})))\}))))).$$

Lastly, the trajectory 4 shown in part (g) of Figure A10 includes Cases 41 to 48 below:

$$\text{Case 41} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\phi^{(t)}(\mathbf{w}_j^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)})))\})))$$

$$\text{Case 42} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\phi^{(t)}(\mathbf{w}_j^{(t)}, \boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_j^{(t)}))))\})))$$

$$\text{Case 43} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\phi^{(t)}(\mathbf{w}_j^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)}))))\})))$$

$$\text{Case 44} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\phi^{(t)}(\mathbf{w}_j^{(t)}, \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_j^{(t)})))))\})))$$

$$\text{Case 45} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_j^{(t)}), \boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)})))\})))$$

$$\text{Case 46} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_j^{(t)}), \boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_j^{(t)}))))\})))$$

$$\text{Case 47} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_j^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_j^{(t)}))))\})))$$

$$\text{Case 48} : \mathbf{w}_i^{(t-1)} = f_i(\psi^{(t)}(\mathcal{A}(\{g_j(\mathbf{w}_j^{(t)})\}), \mathcal{A}(\{g_j(\phi^{(t)}(\mathcal{F}_i(\mathbf{w}_j^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(\mathcal{F}_i(\mathbf{w}_j^{(t)})))))\}))).$$

### A7.3 Canonical Variable Denoising Process

Here, we present all possible canonical variable denoising processes. Due to the absence of noise prediction in the canonical space, a process first *redirects* canonical variable $\mathbf{z}^{(t)}$ to the instance spaces where a subsequence of operations $\boldsymbol{\epsilon}_\theta(\cdot)$, $\phi^{(t)}(\cdot,\cdot)$ and $\psi^{(t)}(\cdot,\cdot)$ are computed.

We exclude the application of $\mathcal{F}_i$ to $\mathbf{w}_i^{(t)} \leftarrow f_i(\mathbf{z}^{(t)})$, as the variable remains unchanged after the operation. Therefore, applying $\mathcal{F}_i$ to $\mathbf{w}_i^{(t)} \leftarrow f_i(\mathbf{z}^{(t)})$ for the inputs of $\boldsymbol{\epsilon}_\theta(\cdot)$, $\phi^{(t)}(\cdot,\cdot)$ and $\psi^{(t)}(\cdot,\cdot)$ is not considered.

Case 4 which belongs to the trajectory 3, is visualized in part (f) of Figure A10. Case 5 and Case 49 derive from the trajectory 4 which are shown in part (h) of Figure A10.

$$\text{Case 49} : \mathbf{z}^{(t-1)} = \psi^{(t)}(\mathbf{z}^{(t)}, \mathcal{A}(\{g_i(\phi^{(t)}(f_i(\mathbf{z}^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)}))))\})))$$

In the trajectory 1, $2^2 = 4$ cases are possible, as shown in part (b) of Figure A10. This includes Case 6 along with Cases 50 to 52 below:

Case 50 : $\mathbf{z}^{(t-1)} = \mathcal{A}(\{g_i(\psi^{(t)}(f_i(\mathbf{z}^{(t)}), \phi^{(t)}(f_i(\mathbf{z}^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)}))))))\})$

Case 51 : $\mathbf{z}^{(t-1)} = \mathcal{A}(\{g_i(\psi^{(t)}(f_i(\mathbf{z}^{(t)}), \mathcal{F}_i(\phi^{(t)}(f_i(\mathbf{z}^{(t)}), \boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)}))))))\})$

Case 52 : $\mathbf{z}^{(t-1)} = \mathcal{A}(\{g_i(\psi^{(t)}(f_i(\mathbf{z}^{(t)}), \mathcal{F}_i(\phi^{(t)}(f_i(\mathbf{z}^{(t)}), \mathcal{F}_i(\boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)}))))))))\}).$

Lastly, trajectory 2, shown in part (d) of Figure A10, encompasses one possible case, corresponding to Case 53:

Case 53 : $\mathbf{z}^{(t-1)} = \mathcal{A}(\{g_i(\psi^{(t)}(f_i(\mathbf{z}^{(t)}), f_i(\phi^{(t)}(\mathbf{z}^{(t)}, \mathcal{A}(\{g_i(\boldsymbol{\epsilon}_\theta(f_i(\mathbf{z}^{(t)})))\}))))))\}).$

### A7.4 Combined Variable Denoising Process

In this section, we introduce combined variable denoising processes where both instance and canonical variables are denoised. This process synchronizes instance variables and a canonical variable by aggregating the unprojected instance variables and the canonical variable in the canonical space.

For clarity, we introduce additional operations below. $\boldsymbol{\delta}_{\mathcal{Z}}(\cdot)$ takes a variable in the canonical space $\mathbf{z} \in \mathcal{Z}$, projects it into the instance spaces, predicts noises in those spaces, and aggregates them back in the canonical space after the unprojection. $\boldsymbol{\Phi}^{(t)}_{\mathcal{Z}}(\cdot)$ then computes Tweedie's formula [43] based on the noise term computed by $\boldsymbol{\delta}_{\mathcal{Z}}(\cdot)$.

$$\boldsymbol{\delta}_{\mathcal{Z}}(\mathbf{z}) = \mathcal{A}(\{g_i(\boldsymbol{\epsilon}_\theta(f_i(\mathbf{z})))\}) \tag{12}$$

$$\boldsymbol{\Phi}^{(t)}_{\mathcal{Z}}(\mathbf{z}) = \phi^{(t)}(\mathbf{z}, \boldsymbol{\delta}(\mathbf{z})). \tag{13}$$

Similarly, given a set of variables in the instance spaces $\{\mathbf{w}_i\}$, the following operators aggregate the unprojected outputs of $\psi^{(t)}(\cdot,\cdot)$, $\boldsymbol{\epsilon}_\theta(\cdot)$ and $\phi^{(t)}(\cdot,\cdot)$ in the canonical space:

$$\boldsymbol{\Psi}^{(t)}_{\mathcal{W}_i}(\{\mathbf{w}_i\}) = \mathcal{A}(\{g_i(\psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))))\}) \tag{14}$$

$$\boldsymbol{\delta}_{\mathcal{W}_i}(\{\mathbf{w}_i\}) = \mathcal{A}(\{g_i(\boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)}))\}) \tag{15}$$

$$\boldsymbol{\Phi}^{(t)}_{\mathcal{W}_i}(\{\mathbf{w}_i\}) = \mathcal{A}(\{g_i(\phi^{(t)}(\mathbf{w}_i^{(t)}, \boldsymbol{\epsilon}_\theta(\mathbf{w}_i^{(t)})))\}) \tag{16}$$

We present joint variable denoising cases on the representative cases discussed in Section A7:

Case 54 : $\mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, \phi^{(t)}(\mathbf{w}_i^{(t)}, f_i(\mathcal{A}(\{\boldsymbol{\delta}_{\mathcal{W}_i}(\{\mathbf{w}_i^{(t)}\}), \boldsymbol{\delta}_{\mathcal{Z}}(\mathbf{z}^{(t)})\}))))$

Case 55 : $\mathbf{w}_i^{(t-1)} = \psi^{(t)}(\mathbf{w}_i^{(t)}, f_i(\mathcal{A}(\{\boldsymbol{\Phi}^{(t)}_{\mathcal{W}_i}(\{\mathbf{w}_i^{(t)}\}), \boldsymbol{\Phi}^{(t)}_{\mathcal{Z}}(\mathbf{z}^{(t)})\})))$

Case 56 : $\mathbf{w}_i^{(t-1)} = f_i(\mathcal{A}(\{\boldsymbol{\Psi}^{(t)}_{\mathcal{W}_i}(\{\mathbf{w}_i^{(t)}\}), \mathbf{z}^{(t-1)}\}))$

Case 57 : $\mathbf{z}^{(t-1)} = \psi^{(t)}(\mathbf{z}^{(t)}, \phi^{(t)}(\mathbf{z}^{(t)}, \mathcal{A}(\{\boldsymbol{\delta}_{\mathcal{Z}}(\mathbf{z}^{(t)}), \boldsymbol{\delta}_{\mathcal{W}_i}(\{\mathbf{w}_i^{(t)}\})\})))$

Case 58 : $\mathbf{z}^{(t-1)} = \psi^{(t)}(\mathbf{z}^{(t)}, \mathcal{A}(\{\boldsymbol{\Phi}^{(t)}_{\mathcal{W}_i}(\{f_i(\mathbf{z}^{(t)})\}), \boldsymbol{\Phi}^{(t)}_{\mathcal{W}_i}(\{\mathbf{w}_i^{(t)}\})\})$

Case 59 : $\mathbf{z}_{t-1} = \mathcal{A}(\{\boldsymbol{\Psi}^{(t)}_{\mathcal{W}_i}(\{f_i(\mathbf{z}^{(t)})\}), \mathcal{A}(\{g_i(\mathbf{w}_i^{(t-1)})\})\}).$

Cases 54 to 59 correspond to the combined variable denoising processes from Cases 1 to 6, respectively. In each of the above cases, we highlight the terms already present in the original representative case in orange and newly added variable to be synchronized together in purple.

### A7.5 Quantitative Results

In Table A10, we present the quantitative results of the 60 diffusion synchronization processes listed above. We follow the same toy experiment setup described in both Section 3.4.1 and Section A2.1. As outlined in Section A7.1, for all instance variable denoising processes, including the independent denoising case (No Synchronization), we perform the final synchronization at the end of the denoising process. For $n$-to-1 projection, we utilize $M = 10$ multiplane images as done in Section 3.4.3.

We report the quantitative results of all cases in Table A10. The results align with the observations of Table 1. In the 1-to-1 projection scenario, most diffusion synchronization processes exhibit similar

performances. Except for Case 55 and Case 56, the combined variable denoising processes (Cases 54 to 59) show suboptimal performances with FID [19] scores over 100. This indicates that denoising either instance variables or a canonical variable is sufficient to produce satisfactory and consistent results.

When it comes to the 1-to-$n$ projection scenario, Case 2 and Case 5 outperform the others, with some exceptions such as Case 11 and Case 35. This trend is also consistent with the results in Section 3.4.1, highlighting the effectiveness of synchronizing the outputs of Tweedie's formula [43] $\phi^{(t)}(\cdot, \cdot)$ even when compared to more complex diffusion synchronization processes.

Lastly, in the $n$-to-1 projection scenario, Case 2 (`SyncTweedies`) is the only one that outperforms the others across all metrics.

In conclusion, as shown in Table A10, Case 2 (`SyncTweedies`) distinctly exhibits superior performance across various projection scenarios, outperforming even more convoluted cases.

Table A10: **A quantitative comparison of all cases in ambiguous image generation.** KID [5] is scaled by $10^3$. For each column, we highlight the row whose value is within 95% of the best.

| | CLIP-A [39] ↑ | CLIP-C [39] ↑ | FID [19] ↓ | KID [5] ↓ |
|---|---|---|---|---|
| | | 1-to-1 Projection | | |
| No Sync. | 28.49 | 62.0 | 102.14 | 51.72 |
| Case 1 | 30.26 | 64.45 | 85.55 | 31.95 |
| **Case 2** (`SyncTweedies`) | 30.35 | 64.52 | 85.36 | 31.82 |
| Case 3 | 30.34 | 64.46 | 85.31 | 31.57 |
| Case 4 | 30.28 | 64.47 | 85.37 | 32.44 |
| Case 5 | 30.36 | 64.43 | 85.56 | 32.1 |
| Case 6 | 30.3 | 64.49 | 84.75 | 31.29 |
| Case 7 | 30.33 | 64.51 | 84.6 | 31.93 |
| Case 8 | 30.27 | 64.49 | 85.03 | 32.08 |
| Case 9 | 29.46 | 62.17 | 97.48 | 44.21 |
| Case 10 | 30.36 | 64.68 | 84.79 | 31.44 |
| Case 11 | 30.31 | 64.48 | 85.81 | 32.26 |
| Case 12 | 30.31 | 64.53 | 84.48 | 31.56 |
| Case 13 | 30.33 | 64.46 | 85.83 | 32.35 |
| Case 14 | 30.33 | 64.57 | 85.69 | 32.36 |
| Case 15 | 30.35 | 64.63 | 85.6 | 32.17 |
| Case 16 | 30.34 | 64.57 | 85.9 | 32.55 |
| Case 17 | 30.32 | 64.5 | 85.66 | 32.3 |
| Case 18 | 30.31 | 64.63 | 85.48 | 32.35 |
| Case 19 | 30.33 | 64.53 | 85.18 | 31.38 |
| Case 20 | 29.91 | 63.48 | 92.18 | 38.44 |
| Case 21 | 30.3 | 64.41 | 85.54 | 32.18 |
| Case 22 | 30.33 | 64.61 | 85.99 | 32.41 |
| Case 23 | 30.31 | 64.59 | 85.17 | 31.77 |
| Case 24 | 30.06 | 63.91 | 91.82 | 37.62 |
| Case 25 | 30.3 | 64.46 | 85.41 | 32.22 |
| Case 26 | 30.36 | 64.59 | 84.98 | 31.93 |
| Case 27 | 30.31 | 64.49 | 84.89 | 31.8 |
| Case 28 | 30.33 | 64.42 | 85.34 | 32.61 |
| Case 29 | 30.33 | 64.55 | 85.93 | 32.29 |
| Case 30 | 30.33 | 64.51 | 85.03 | 31.72 |
| Case 31 | 30.32 | 64.42 | 85.95 | 32.91 |
| Case 32 | 30.33 | 64.5 | 85.78 | 32.35 |
| Case 33 | 30.34 | 64.63 | 85.77 | 32.4 |
| Case 34 | 30.37 | 64.66 | 84.99 | 31.84 |
| Case 35 | 30.36 | 64.59 | 85.39 | 31.64 |
| Case 36 | 30.34 | 64.55 | 84.59 | 31.8 |
| Case 37 | 30.33 | 64.63 | 85.21 | 31.84 |
| Case 38 | 30.39 | 64.56 | 84.75 | 31.82 |
| Case 39 | 30.31 | 64.55 | 85.56 | 32.67 |
| Case 40 | 30.29 | 64.55 | 85.44 | 32.17 |
| Case 41 | 30.31 | 64.48 | 85.53 | 32.02 |
| Case 42 | 30.35 | 64.47 | 85.62 | 32.68 |
| Case 43 | 30.31 | 64.55 | 85.4 | 32.09 |
| Case 44 | 30.3 | 64.51 | 86.13 | 32.55 |
| Case 45 | 30.32 | 64.42 | 85.44 | 32.25 |
| Case 46 | 30.34 | 64.51 | 85.59 | 32.67 |
| Case 47 | 30.32 | 64.52 | 85.06 | 31.76 |
| Case 48 | 30.35 | 64.57 | 84.95 | 31.96 |
| Case 49 | 30.3 | 64.48 | 85.46 | 32.43 |
| Case 50 | 30.31 | 64.46 | 86.49 | 32.97 |
| Case 51 | 30.3 | 64.47 | 85.83 | 32.41 |

| | CLIP-A [39] ↑ | CLIP-C [39] ↑ | FID [19] ↓ | KID [5] ↓ |
|---|---|---|---|---|
| Case 52 | 30.35 | 64.62 | 86.05 | 32.44 |
| Case 53 | 30.28 | 64.45 | 85.38 | 32.21 |
| Case 54 | 21.27 | 50.03 | 422.05 | 491.69 |
| Case 55 | 30.09 | 64.26 | 87.04 | 34.26 |
| Case 56 | 30.33 | 64.4 | 87.26 | 33.06 |
| Case 57 | 21.28 | 49.99 | 420.29 | 495.9 |
| Case 58 | 27.02 | 60.08 | 113.68 | 57.55 |
| Case 59 | 26.88 | 60.05 | 116.9 | 60.24 |
| 1-to-$n$ Projection | | | | |
| No Sync. | 27.64 | 56.97 | 132.6 | 107.41 |
| Case 1 | 26.12 | 54.97 | 231.23 | 212.39 |
| **Case 2** (`SyncTweedies`) | 30.23 | 60.87 | 110.73 | 76.25 |
| Case 3 | 29.96 | 60.49 | 118.53 | 86.38 |
| Case 4 | 25.86 | 54.29 | 254.74 | 250.76 |
| Case 5 | 30.29 | 61.0 | 108.77 | 74.51 |
| Case 6 | 30.0 | 60.55 | 117.64 | 84.84 |
| Case 7 | 28.41 | 58.41 | 195.37 | 186.39 |
| Case 8 | 25.79 | 54.21 | 271.32 | 278.01 |
| Case 9 | 28.77 | 57.56 | 129.27 | 99.92 |
| Case 10 | 30.11 | 60.91 | 115.29 | 84.28 |
| Case 11 | 30.2 | 60.84 | 110.38 | 76.53 |
| Case 12 | 29.92 | 60.8 | 121.93 | 86.54 |
| Case 13 | 29.93 | 60.84 | 121.64 | 86.13 |
| Case 14 | 29.3 | 59.91 | 163.26 | 127.25 |
| Case 15 | 29.53 | 60.29 | 158.83 | 140.3 |
| Case 16 | 30.05 | 60.45 | 117.48 | 86.86 |
| Case 17 | 30.1 | 60.74 | 118.56 | 88.96 |
| Case 18 | 30.18 | 60.77 | 113.1 | 79.59 |
| Case 19 | 30.17 | 60.79 | 117.68 | 85.37 |
| Case 20 | 29.45 | 59.41 | 119.8 | 87.59 |
| Case 21 | 30.06 | 60.76 | 115.39 | 82.83 |
| Case 22 | 30.02 | 60.58 | 116.08 | 83.42 |
| Case 23 | 30.06 | 60.69 | 117.55 | 84.91 |
| Case 24 | 29.45 | 59.43 | 121.43 | 89.89 |
| Case 25 | 29.94 | 60.45 | 118.35 | 86.34 |
| Case 26 | 30.02 | 60.55 | 119.02 | 86.96 |
| Case 27 | 29.92 | 60.45 | 118.86 | 86.83 |
| Case 28 | 30.01 | 60.52 | 119.1 | 86.91 |
| Case 29 | 29.94 | 60.54 | 118.76 | 85.98 |
| Case 30 | 29.97 | 60.52 | 120.49 | 89.38 |
| Case 31 | 29.95 | 60.45 | 119.19 | 86.53 |
| Case 32 | 30.02 | 60.62 | 119.03 | 86.73 |
| Case 33 | 29.95 | 60.52 | 119.92 | 87.55 |
| Case 34 | 29.96 | 60.54 | 119.57 | 87.29 |
| Case 35 | 30.2 | 60.84 | 110.38 | 76.08 |
| Case 36 | 29.92 | 60.8 | 121.93 | 86.99 |
| Case 37 | 29.94 | 60.45 | 118.35 | 86.27 |
| Case 38 | 30.02 | 60.55 | 119.02 | 86.99 |
| Case 39 | 29.94 | 60.45 | 118.35 | 86.05 |
| Case 40 | 30.02 | 60.55 | 119.02 | 87.07 |
| Case 41 | 29.92 | 60.45 | 118.86 | 86.51 |
| Case 42 | 30.01 | 60.52 | 119.1 | 86.46 |
| Case 43 | 29.94 | 60.54 | 118.76 | 86.42 |
| Case 44 | 29.97 | 60.52 | 120.49 | 89.04 |
| Case 45 | 29.95 | 60.45 | 119.19 | 86.38 |
| Case 46 | 30.02 | 60.62 | 119.03 | 87.21 |
| Case 47 | 29.95 | 60.52 | 119.92 | 87.91 |
| Case 48 | 29.96 | 60.54 | 119.57 | 87.2 |
| Case 49 | 29.32 | 59.98 | 162.41 | 126.53 |
| Case 50 | 30.0 | 60.62 | 118.17 | 85.47 |
| Case 51 | 29.96 | 60.53 | 119.6 | 87.85 |
| Case 52 | 30.02 | 60.62 | 119.79 | 87.78 |
| Case 53 | 30.0 | 60.62 | 118.17 | 85.66 |
| Case 54 | 21.25 | 49.99 | 442.91 | 538.91 |
| Case 55 | 25.99 | 55.43 | 223.48 | 200.81 |
| Case 56 | 25.9 | 55.22 | 229.84 | 210.01 |
| Case 57 | 21.25 | 50.02 | 423.48 | 501.66 |
| Case 58 | 28.21 | 58.01 | 151.01 | 122.97 |
| Case 59 | 28.05 | 57.96 | 152.2 | 123.95 |
| $n$-to-1 Projection | | | | |
| No Sync. | 28.26 | 61.75 | 111.11 | 57.24 |
| Case 1 | 21.28 | 49.94 | 405.82 | 496.98 |
| **Case 2** (`SyncTweedies`) | 29.56 | 63.1 | 96.3 | 40.91 |
| Case 3 | 21.58 | 50.58 | 243.23 | 151.11 |
| Case 4 | 21.33 | 50.05 | 301.2 | 233.11 |
| Case 5 | 21.09 | 50.04 | 289.82 | 213.45 |

| | CLIP-A [39] ↑ | CLIP-C [39] ↑ | FID [19] ↓ | KID [5] ↓ |
|---|---|---|---|---|
| Case 6 | 22.28 | 50.11 | 329.11 | 299.11 |
| Case 7 | 21.76 | 50.01 | 422.31 | 518.33 |
| Case 8 | 21.72 | 50.0 | 426.69 | 530.85 |
| Case 9 | 22.81 | 51.54 | 192.3 | 112.63 |
| Case 10 | 25.05 | 56.05 | 158.99 | 92.29 |
| Case 11 | 25.67 | 54.72 | 288.88 | 278.59 |
| Case 12 | 25.67 | 56.48 | 160.32 | 92.54 |
| Case 13 | 25.11 | 53.61 | 260.91 | 223.48 |
| Case 14 | 24.29 | 52.09 | 344.55 | 379.52 |
| Case 15 | 25.38 | 53.72 | 259.18 | 221.03 |
| Case 16 | 27.93 | 58.75 | 198.49 | 144.28 |
| Case 17 | 25.02 | 53.69 | 194.6 | 130.03 |
| Case 18 | 25.32 | 53.76 | 315.92 | 329.26 |
| Case 19 | 24.65 | 53.49 | 212.36 | 157.81 |
| Case 20 | 21.53 | 50.71 | 236.09 | 157.04 |
| Case 21 | 22.47 | 52.48 | 189.73 | 104.92 |
| Case 22 | 23.74 | 54.67 | 154.53 | 77.4 |
| Case 23 | 22.75 | 53.44 | 174.95 | 87.28 |
| Case 24 | 21.63 | 50.83 | 206.25 | 110.93 |
| Case 25 | 24.72 | 57.52 | 130.55 | 60.76 |
| Case 26 | 21.53 | 50.81 | 211.99 | 122.67 |
| Case 27 | 21.53 | 50.4 | 246.63 | 161.28 |
| Case 28 | 21.44 | 50.9 | 253.72 | 157.43 |
| Case 29 | 21.28 | 50.75 | 249.3 | 151.45 |
| Case 30 | 21.58 | 50.84 | 249.85 | 152.37 |
| Case 31 | 21.69 | 50.63 | 233.72 | 144.35 |
| Case 32 | 21.89 | 50.68 | 243.65 | 160.12 |
| Case 33 | 22.22 | 51.04 | 208.49 | 117.23 |
| Case 34 | 22.01 | 50.48 | 257.02 | 171.18 |
| Case 35 | 25.72 | 54.78 | 289.98 | 279.62 |
| Case 36 | 25.73 | 56.58 | 160.42 | 93.2 |
| Case 37 | 24.79 | 57.46 | 131.86 | 61.08 |
| Case 38 | 21.52 | 50.86 | 211.71 | 121.06 |
| Case 39 | 24.77 | 57.45 | 130.03 | 59.89 |
| Case 40 | 21.58 | 50.88 | 212.99 | 122.21 |
| Case 41 | 21.52 | 50.5 | 247.4 | 161.85 |
| Case 42 | 21.46 | 50.86 | 253.6 | 157.02 |
| Case 43 | 21.31 | 50.67 | 249.7 | 151.85 |
| Case 44 | 21.54 | 50.85 | 251.18 | 154.49 |
| Case 45 | 21.65 | 50.54 | 237.56 | 147.55 |
| Case 46 | 21.94 | 50.69 | 244.4 | 161.12 |
| Case 47 | 22.27 | 51.05 | 206.73 | 114.38 |
| Case 48 | 22.05 | 50.47 | 258.46 | 174.1 |
| Case 49 | 21.13 | 50.05 | 280.71 | 195.28 |
| Case 50 | 22.73 | 50.03 | 350.36 | 322.78 |
| Case 51 | 22.29 | 50.04 | 354.31 | 332.85 |
| Case 52 | 22.31 | 50.06 | 349.66 | 322.96 |
| Case 53 | 22.74 | 50.06 | 349.41 | 321.65 |
| Case 54 | 20.77 | 50.06 | 419.46 | 495.55 |
| Case 55 | 22.01 | 50.15 | 270.14 | 192.31 |
| Case 56 | 21.8 | 50.1 | 291.59 | 217.05 |
| Case 57 | 21.56 | 50.03 | 405.75 | 477.82 |
| Case 58 | 26.32 | 58.49 | 124.28 | 66.34 |
| Case 59 | 26.52 | 59.09 | 108.41 | 46.07 |