DreamPropeller: Supercharge Text-to-3D Generation with Parallel Sampling

Linqi Zhou[1*] Andy Shih[1] Chenlin Meng[1,2] Stefano Ermon[1]
[1]Stanford University, [2]Pika Labs
[1]{linqizhou, andyshih, chenlin, ermon}@stanford.edu, [2]chenlin@pika.art

Abstract

Recent methods such as Score Distillation Sampling (SDS) and Variational Score Distillation (VSD) using 2D diffusion models for text-to-3D generation have demonstrated impressive generation quality. However, the long generation time of such algorithms significantly degrades the user experience. To tackle this problem, we propose DreamPropeller, a drop-in acceleration algorithm that can be wrapped around any existing text-to-3D generation pipeline based on score distillation. Our framework generalizes Picard iterations, a classical algorithm for parallel sampling an ODE path, and can account for non-ODE paths such as momentum-based gradient updates and changes in dimensions during the optimization process as in many cases of 3D generation. We show that our algorithm trades parallel compute for wallclock time and empirically achieves up to 4.7x speedup with a negligible drop in generation quality for all tested frameworks. Our implementation can be found here.

1. Introduction

Diffusion models [9, 38–40] have seen great success in a variety of domains, including both 2D images [22, 31, 49, 49] and 3D shapes [21, 37, 49, 49, 50]. Early attempts of applying diffusion frameworks to 3D shape generation require training separate models for each data category with 3D training samples. This problem is significantly alleviated by the use of large-scale text-conditioned 2D diffusion models [31]. Such large-scale foundation models, often trained on internet-scale data such as LAION-5B [35], have exhibited remarkable semantic understanding of the visual world, which has inspired their use for creating 3D shapes in open-vocabulary settings. Among the most remarkable developments in recent years involves optimizing NeRF [23] scenes using Score Distillation Sampling (SDS) [29] or Score Jacobian Chaining (SJC) [45], which push the 2D rendering of the 3D models to be more likely under the 2D diffusion prior through gradient-based optimization. More recent developments including Variational Score Distillation (VSD) [46] further improve the quality of 3D shape generation.

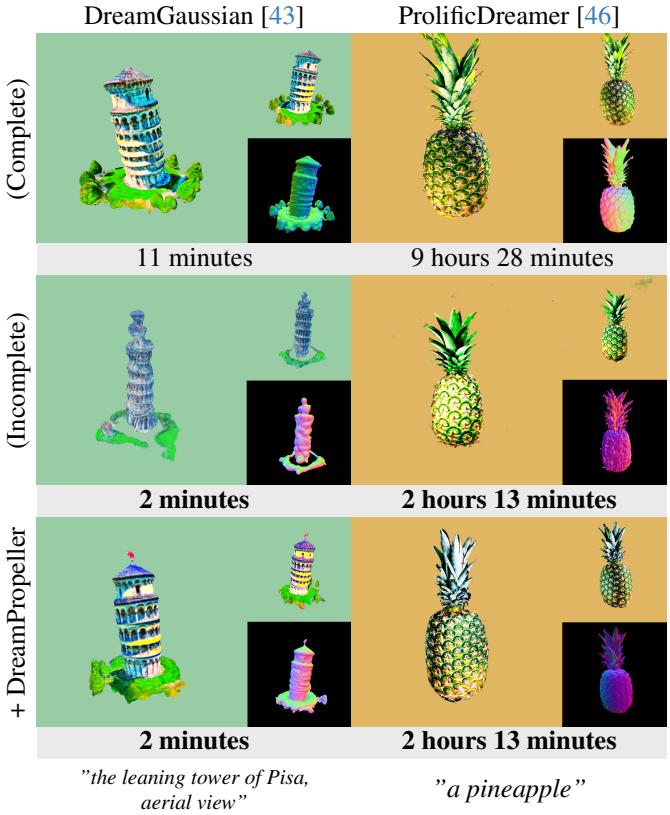


Figure 1. We present two representative examples of applying DreamPropeller. Gray rows denote runtime. Our framework trades parallel compute for speed and achieves more than 4x speedup when applied to both DreamGaussian [43][1] and ProlificDreamer [46] while maintaining the generation quality. At the time when DreamPropeller finishes, the baseline versions (Incomplete) exhibit significantly worse appearance and geometry.

Despite promising results in generation quality, text-to-3D methods using SDS/VSD suffer from long generation time because gradient-based optimization is computation-

*Work done at Pika Labs.
[1]We use batch size 16 for higher-quality generation, which requires longer time.

ally expensive over the parameters of the 3D representation, and a large number of iterations are required for convergence. Recent works [5, 43, 48] have explored using a novel efficient 3D representation, namely 3D Gaussian Splatting [11], to accelerate the distillation process. Others have explored amortizing the generation process by distilling a text-conditioned generator that can generate 3D shapes at test time in one step [20]. Orthogonal to these methods, we propose a general framework to accelerate text-to-3D creation (regardless of the underlying 3D representation) by leveraging parallel compute to perform the same number of optimization steps faster (in terms of wall-clock time).

Our method is based on a generalization of Picard iterations [1] which we adapt to text-to-3D generation. While Picard iterations were previously shown to be effective in accelerating image generation from diffusion models through faster (parallel-in-time) ODE solving [36], we generalize the approach to more complex computation graphs involving multiple gradient updates and changes in the dimensions of the variables. We show that all existing 3D representations for text-to-3D generation can be accelerated by our method, including the recently proposed 3D Gaussian Splatting. Experimentally, we also achieve 4.7x times speedup with negligible degradation in generation quality.

2. Related Works

Text-to-Image diffusion models. Diffusion models [9, 38–40] rose to popularity due to their superior performance and training stability compared to GANs [8]. They have been scaled up to large-scale foundation models, among the most popular of which are text-conditioned latent-diffusion models (LDM) [3, 31, 34], which are built on the latent space of an autoencoder.

Such text-conditional LDMs, often trained on internet-scale datasets [35], can achieve high generation fidelity with reasonable faithfulness to input prompts. This has inspired many other remarkable applications such as adding additional control signals [49], customizing for subject-specific generation [7, 13, 19, 32], and editing [22, 25]. The utilization of large-scale Text-to-Image diffusion models lies at the core of modern visual generative modeling, and is also central to our method for text-to-3D generation.

Text/Image-to-3D generation via score distillation. DreamFusion [29] and Score Jacobian Chaining [45] were among the first to propose lifting 2D diffusion models for 3D generation by Score Distillation Sampling (SDS), which propagates the score of pretrained diffusion models to the differentiable rendering of NeRF [23]. ProlificDreamer [46] introduces Variational Score Distillation (VSD) which significantly improves the quality of 3D generation by adaptively training a LoRA model. Orthogonal to the distillation algorithm, other works such as [4, 15] propose to optimize a more memory-efficient DMTet representation for high-resolution rendering. A more recent work [44] proposes to volumetrically render signed distance fields (SDF) for better texture and mesh extraction. Others [5, 43, 48] have also investigated the use of 3D Gaussian Splatting [11] as the underlying representation for fast and efficient generation, bringing the creation time down to as low as two minutes. Besides other applications such as controllable scene composition [2, 6, 28], SDS is also widely applied in the Image-to-3D task, where Zero-1-to-3 [17] finetunes a view-dependent diffusion model for 3D generation given single images. Other works [16, 30, 47] further improved image-conditioned generation by integrating multi-view information and additional guidance into the generation process.

Accelerating sampling with parallelism. Many works have studied accelerated sampling of generative models by leveraging parallel computation to trade compute for speed. For autoregressive models, prior works have used Jacobi/Gauss iteration [41] or predict/accept mechanisms [42] to improve sampling speed. For diffusion models, parallelism based on fixed-point iterations has been shown to speed up sampling of pretrained diffusion models [36]. Similarly, the goal of our work is to accelerate sampling, specifically for 3D generation models, which are known to have slow sampling speed. Our technique is most related to the work of Shih et al. [36], which is inspired by the classic technique of Picard iterations to solve ODEs using parallel computation, allowing for the utilization of multiple GPUs to accelerate sampling. Unfortunately, the method of Picard iteration cannot be directly used to parallelize the sequential gradient update steps of 3D generation, because of the use of momentum-based gradient updates changes in dimension during optimization.

In this work, we seek to overcome the challenges of the representational differences of 3D generation and design an algorithm that accelerates the generation of all existing 3D representations. We do so by formulating Picard iterations [36] for a wider family of sequential computation, enabling us to leverage parallel computational resources to accelerate 3D generation.

3. Preliminary

Recent Text-to-3D generation frameworks mostly rely on distilling knowledge from pretrained 2D diffusion models. We hereby introduce some backgrounds and notations helpful for our exposition.

3.1. Text-to-3D Generation via Score Distillation

2D diffusion models [9, 38–40] learn a distribution of images by adding noise to the ground-truth image $\mathbf{x}$ at noise level t, resulting in noisy images $\mathbf{z}_t$, and uses a score network $\boldsymbol{\epsilon}_\phi(\mathbf{z}_t, t, y)$ with parameter ϕ and caption y to estimate the gradient direction towards higher likelihood given

$\mathbf{z}_t$. Such diffusion models can faithfully produce realistic images highly aligned with the input caption y. Their success has also inspired their use for 3D shape generation, which we shall introduce below. Suppose the 3D shape (*e.g.* NeRF) parameterized by θ, can be rendered into an image following a deterministic transformation $\boldsymbol{g} : \Theta \times \mathcal{C} \to \mathbb{R}^{H \times W \times C}$ which takes in shape parameter $\theta \in \Theta$ and camera parameters $c \in \mathcal{C}$. We seek an update rule for θ such that $\boldsymbol{g}(\theta, c)$ is a realistic image following caption y.

Score Distillation Sampling. The seminal works [29, 45] on lifting 2D diffusion models for 3D creation propose to directly propagate diffusion score prediction towards NeRF parameters. The most prominent algorithm, Score Distillation Sampling (SDS), or Score Jacobian Chaining (SJC), states that a 3D model parameterized by can be guided to generate a scene with caption y by following the update rule $\theta_{\tau+1} = \theta_\tau - \eta \nabla_{\theta_\tau} \mathcal{L}_{\text{SDS}}$. With $t \sim \mathcal{U}(0.02, 0.98)$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and $\mathbf{z}_t = \alpha_t \boldsymbol{g}(\theta, c) + \sigma_t \boldsymbol{\epsilon}$,

$$\nabla_\theta \mathcal{L}_{\text{SDS}} = \mathbb{E}_{t,\boldsymbol{\epsilon}} \left[\omega(t) \Big(\boldsymbol{\epsilon}_{\text{pretrain}}(\mathbf{z}_t, t, y) - \boldsymbol{\epsilon} \Big) \frac{\partial}{\partial \theta} \boldsymbol{g}(\theta, c) \right] \tag{1}$$

where $\omega(t)$ is a weighting function. In practice, the gradient update is done through Adam gradient update, which updates 3D models such that their rendering $\boldsymbol{g}(\theta, c)$ closely follows distributions of the pretrained 2D diffusion prior.

Variational Score Distillation. Despite success in zero-shot 3D NeRF generation, SDS often suffers from over-saturation and simplistic geometry. To enhance the generative quality, another recent work [46] proposes Variational Score Distillation (VSD), which replaces the noise sample $\boldsymbol{\epsilon}$ with a trainable LoRA diffusion with parameter ϕ such that

$$\nabla_\theta \mathcal{L}_{\text{VSD}} = \mathbb{E}_{t,\boldsymbol{\epsilon}} \Big[\omega(t) \Big(\boldsymbol{\epsilon}_{\text{pretrain}}(\mathbf{z}_t, t, y) - \boldsymbol{\epsilon}_\phi(\mathbf{z}_t, t, c, y) \Big) \frac{\partial}{\partial \theta} \boldsymbol{g}(\theta, c) \Big] \tag{2}$$

where the LoRA model is adaptively trained to fit the distribution of the current render $\boldsymbol{g}(\theta, c)$.

Generative 3D Gaussian Splatting. 3D Gaussian Splatting [11] is a recently developed 3D representation for efficient rendering. It is represented by a set of 3D Gaussians which are optimized with differentiable rasterizers. More recent works [5, 43, 48] have adopted this representation for score distillation, which greatly reduces the generation time. However, this representation is different from others in that the number of Gaussians can change during the optimization process due to its split-and-prune operations.

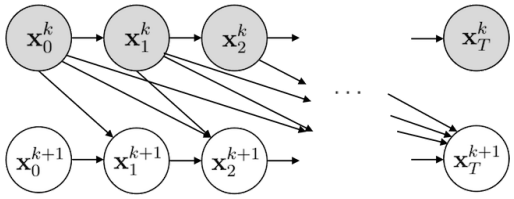


Figure 2. Picard dependency graph. Gray nodes have outgoing edges to all subsequent nodes in $k+1$-th iteration and are independent of each other. This allows parallel computation of $s(\mathbf{x}_\tau^k, \tau)$ for all $\tau \in [0, T-1]$.

3.2. Picard Iterations

The classic Picard iteration approximates the solution trajectory of an ODE $\mathbf{x}_{0:\tau}$ ending at time τ

$$\mathbf{x}_\tau = \mathbf{x}_0 + \int_0^\tau s(\mathbf{x}_u, u) \mathrm{d}u \tag{3}$$

via fixed-point iteration, by starting with a guess of the full trajectory $\mathbf{x}_{0:T}^{k=0}$ and iteratively refining until convergence using the following iteration rule for each iteration k:

$$\mathbf{x}_\tau^k = \mathbf{x}_0^{k-1} + \int_0^\tau s(\mathbf{x}_u^{k-1}, u) \mathrm{d}u \tag{4}$$

which under mild conditions will converge to a unique ODE solution given initial condition $\mathbf{x}_0$. The discrete-time approximation is

$$\mathbf{x}_\tau^k = \mathbf{x}_0^{k-1} + \frac{1}{T} \sum_{i=0}^{\tau-1} s(\mathbf{x}_i^{k-1}, \frac{i}{T}) \tag{5}$$

which is guaranteed to converge in T steps, but in practice often converges much faster in $K \ll T$ iterations. This procedure allows us to leverage parallel computation to converge to the solution in $O(K)$ steps. In Figure 2 we depict the computation graph of Picard iterations, where we see long-range dependencies allowing for information to propagate quickly to the end of the sequence.

Previously, Shih et al. [36] leveraged Picard iterations to accelerate sampling for diffusion models on images. Once converged, we simply extract the endpoint of the solution trajectory of the final iteration $\mathbf{x}_{\tau=T}^{k=K}$ as the sample of the diffusion model.

In this work, we seek to extend the method of Picard iterations to Text-to-3D generation via score distillation. However, generalizing this process to 3D generation is non-trivial because, unlike Shih et al. [36], solution trajectories for 3D generation is not in data space, but rather in "parameter" space. This presents key challenges: 1) for methods such as 3D Gaussian Splatting, the parameter space has changing dimensionality over the length of the trajectory, 2) all score distillation methods involve sequential gradient

update using Adam optimizer, as opposed to a trivial prefix sum. Therefore, we must investigate more deeply to come up with a generalized formulation of Picard iteration that is applicable to 3D generation.

4. Method

Despite the impressive quality of the generation results from VSD and SDS, all of score distillation methods suffer from long generation time (*e.g.* 10 hours for one generation from VSD on an NVIDIA A100), making them prohibitively expensive to use in practice. Motivated by this problem, we seek to design an acceleration algorithm for all existing 3D representations suitable for score distillation.

4.1. Score Distillation as Sequential Computation

Our key insight is that the parameter update rules for SDS/VSD can roughly be written as a sequential computation $\theta_{\tau+1} = \theta_\tau + \eta \nabla_{\theta_\tau} \mathcal{L}$ of a form that is similar to Picard iterations, where $\mathcal{L}$ can either be $\mathcal{L}_{\text{SDS}}$ or $\mathcal{L}_{\text{VSD}}$ and η is step size. However, we cannot directly apply Picard iterations because these methods have further complications in their sequential computations. For example, SDS/VSD rely on momentum-based optimizers such as Adam [12] to perform the parameter update, where the momentum prevents us from directly using vanilla Picard iteration. Special representations such as Gaussian Splatting intertwine gradient updates with splitting operations, which increase the dimensionality of θ, a property that similarly prevents the naïve use of Picard.

4.2. Generalizing Picard Iterations

We now present a generalized version of Picard iterations that will enable us to apply the same parallelization techniques to more complicated computation graphs, encompassing cases such as Gaussian Splatting and SDS/VSD.

In standard Picard iterations, our goal is to parallelize an ODE that takes additive sequential updates of the form

$$\theta_{\tau+1} = g(\theta_\tau) = \theta_\tau + \eta s(\theta_\tau) \tag{6}$$

where $s(\cdot)$ is the drift function of the ODE, and the eventual *computational unit* that we will parallelize over. To take the first step towards generalizing Picard iteration, we rearrange to write this computational unit s explicitly:

$$s(\theta_\tau) = \frac{1}{\eta}(g(\theta_\tau) - \theta_\tau) \tag{7}$$

where $g(\cdot)$ denotes the underlying function outputting the next parameter θ_τ. Choosing the computational unit s as the drift is natural for ODEs, and is convenient because unrolling drifts can be easily done via a summation. However, this choice of computing the drift cannot be generalized to settings such as expanding/shrinking of dimensions and discrete domains, where subtraction operator can be unnatural or even undefined. Moreover, in settings using momentum-based updates (Adam), applying drift-based error accumulation to the momentum terms can lead to very poor performance.

Algorithm 1 Generalized Picard Iteration

Input: Initial parameter θ, pseudo-inverse $h^\dagger$, drift s, maximum time T
Output: Score distillation output
$\theta_\tau^0 \leftarrow \theta, \quad \forall \tau \in [0, T]$
$\tau, k \leftarrow 0, 0$
while not converged **do**
  for τ from 0 to $T-1$ **do** compute $s(\theta_\tau^k)$ in parallel
  for τ from 0 to $T-1$ **do** $\theta_{\tau+1}^{k+1} \leftarrow h^\dagger(s(\theta_\tau^k), \theta_\tau^k)$
  $k \leftarrow k+1$
return θ_T^k

To generalize Picard iterations, our insight is that we can consider many different choices of the computational unit s. We can write s as some general function of $g(\theta_\tau)$ and θ_τ:

$$s(\theta_\tau) = h(g(\theta_\tau), \theta_\tau) \tag{8}$$

where h is equipped with $h^\dagger(\cdot, \cdot)$, a pseudo-inverse of h w.r.t. the first argument $g(\theta_\tau)$. Under this condition, we can write the following iteration rule:

$$\theta_\tau^k = h^\dagger\left(s(\theta_{\tau-1}^{k-1}), h^\dagger\left(s(\theta_{\tau-2}^{k-1}), \dots h^\dagger(s(\theta_0^{k-1}), \theta_0^{k-1})\right)\right) \tag{9}$$

This iteration rule can be understood as the generalized form of Picard iterations. Similar to Picard iterations, we 1) perform the computation units $s(\theta_{0:T}^{k-1})$ in parallel and 2) do a sequential unrolling of the trajectory from iteration $k-1$ to arrive at iteration k. In contrast, Picard iteration has a simple form for the sequential unrolling, *i.e.* cumulative sum (as in Table 1).

We further show in Appendix A that the existence of $h^\dagger$ paired with the iteration rule in Eq. (9) is well-defined and guarantees a fixed-point solution.

Finally, we proceed by demonstrating two concrete choices of s for generalized Picard iterations on 3D Gaussian Splatting and SDS, both of which pose problems for naive Picard iteration.

Example 1: 3D Gaussian Splatting

Recently, 3D Gaussian Splatting [11] has been adopted for 3D generative models using SDS [5, 43, 48]. A unique property of this representation is its ability to dynamically change its dimension (*i.e.* number of Gaussians) during the optimization process. This poses significant challenges to

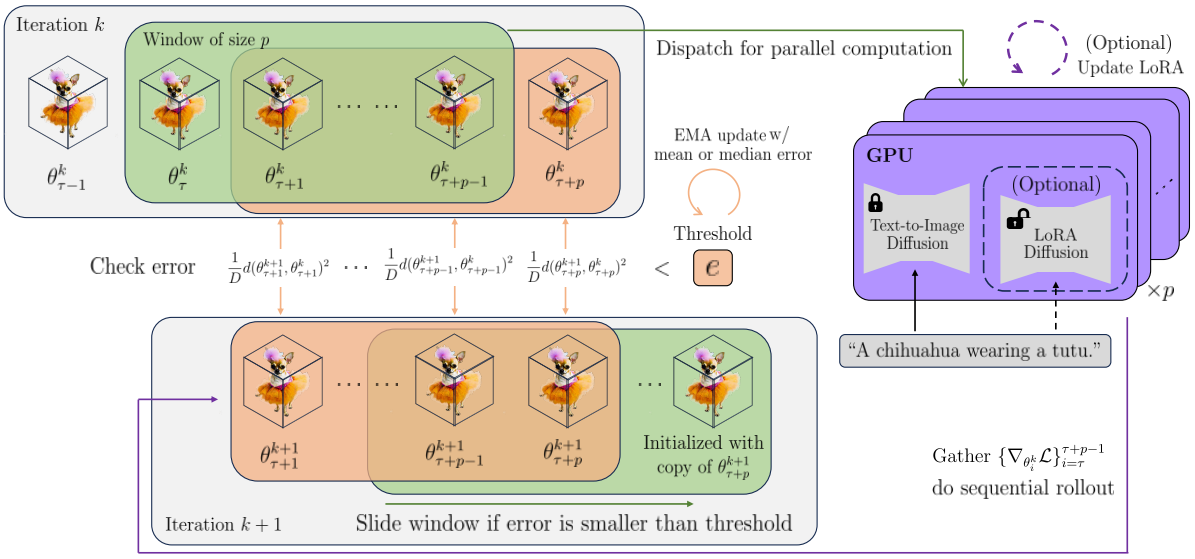


Figure 3. Overview of DreamPropeller. Starting from top left, for iteration k, we initialize a window of 3D shapes (in green) with dimension D and dispatch them to p GPUs for parallelly computing the SDS/VSD gradients, which are gathered for rollout using the rule in Eq. (9). The resulting shapes (in orange) for iteration $k+1$ are compared to those in iteration k. The window is slid forward until the error at that time step is not smaller than the threshold e, which is adaptively updated with the mean/median error of the window. Optionally, in the case of VSD, we keep independent copies of LoRA diffusion on all GPUs which are updated independently without extra communication.

	Picard	Generalized
Sequential output $\theta^{k+1}_{\tau+1}$	$g(\theta^k_\tau)$	$g(\theta^k_\tau)$
Update function $s(\theta^k_\tau)$	$s(\theta^k_\tau) = \frac{1}{\eta}(g(\theta^k_\tau) - \theta^k_\tau)$	$s(\theta^k_\tau) = h(g(\theta^k_\tau), \theta^k_\tau)$
Output parameterization	$\theta^{k+1}_{\tau+1} = \theta^k_\tau + \eta s(\theta^k_\tau)$	$\theta^{k+1}_{\tau+1} = h^\dagger(s(\theta^k_\tau), \theta^k_\tau)$
Iteration rule	$\theta^k_\tau = \theta^{k-1}_0 + \eta \sum_{i=0}^{\tau-1} s(\theta^{k-1}_i)$	$\theta^k_\tau = h^\dagger(s(\theta^{k-1}_{\tau-1}), \dots h^\dagger(s(\theta^{k-1}_0), \theta^{k-1}_0) \dots)$

Table 1. Comparison between Picard and generalized iterations.

the classical Picard iterations as it assumes fixed dimensions throughout (*i.e.* $\theta_\tau, \theta_{\tau+1}$ have the same dimension).

Concretely, in the case of 3D Gaussian Splatting where the number of points may increase, the difference in dimension θ^k_τ and $g(\theta^k_\tau)$ can be addressed by designing $h(g(\theta^k_\tau), \theta^k_\tau)$ to be

$$s(\theta^k_\tau) = \frac{1}{\eta}(\text{proj}(g(\theta^k_\tau)) - \theta^k_\tau) \tag{10}$$

where proj : $\mathbb{R}^N \to \mathbb{R}^M$, $N \geq M$, is a projection function from $\mathbb{R}^N$ to $\mathbb{R}^M$ by deleting a subset of points from its input. Its pseudo-inverse can be

$$\theta^{k+1}_{\tau+1} = \text{unproj}(\theta^k_\tau + \eta s(\theta^k_\tau)) \tag{11}$$

where unproj : $\mathbb{R}^M \to \mathbb{R}^N$ is a dimensionality-increasing function that adds new points to the current parameter. Following Tang et al. [43], we use the split-and-clone function as our unproj$(\cdot)$ function.

Example 2: Adam gradient updates

With methods that use optimizers such as Adam, the sequential update involves a momentum term (here denoted as m^k_τ) which necessitates the use of the generalized form of Picard iterations, since applying vanilla Picard iterations on $s((\theta^k_\tau, m^k_\tau))$ will lead to poor updating of the additional momentum parameters.

To incorporate the additional momentum update rules, we can simply design $h(g(\theta^k_\tau, m^k_\tau), (\theta^k_\tau, m^k_\tau))$ to be $\nabla_{\theta^k_\tau}\mathcal{L}$, which gives us (assuming learning rate η) the natural pseudo-inverse

$$\theta^{k+1}_{\tau+1}, m^{k+1}_{\tau+1} = \text{Adam}(s((\theta^k_\tau, m^k_\tau)), (\theta^k_\tau, m^k_\tau)) \tag{12}$$

and the generalized Picard update becomes

$$\theta^{k+1}_{\tau+1}, m^{k+1}_{\tau+1} \leftarrow g(\theta^k_\tau, m^k_\tau) = \text{Adam}(\nabla_{\theta^k_\tau}\mathcal{L}, (\theta^k_\tau, m^k_\tau)) \tag{13}$$

For clarity, we additionally provide a high-level algorithm in Algorithm 1. Because of parallel computation of $s(\cdot)$, this algorithm can converge in $\boldsymbol{O}(K)$ where $K \ll T$.

4.3. Practical Decisions

We note several practical considerations that are significant for empirical success.

Sliding window. Although one can in theory parallelize the entire trajectory, it is impractical to start by keeping all of $\{\theta_\tau^0\}_{\tau=0}^T$ in memory. In 3D generation settings, T can usually be on the order of 10K and each parameter can cost large amounts of memory. We therefore similarly take inspiration from [36] and employ a batched window scheme such that the Picard iteration is only performed on $\theta_{\tau:\tau+p}^k$ and the window is slid across until the fixed-point convergence error at the starting time step is above a threshold e. More details on the distance metric and how to handle dimension mismatch can be found in Appendix C.

In addition, to maximize efficiency, we want to parallelize the expensive calculation of SDS/VSD gradients, so we need to put one pretrained diffusion model on each GPU. We set the window size to be one less than the total number of GPUs, and use the remaining one for sequential rollout.

Eliminating stochasticity. As Picard iterations' convergence depends on the deterministic fixed-point iteration scheme, which requires deterministic gradient for the same θ_τ^k at each τ. However, the calculation of both $\nabla_{\theta_\tau}\mathcal{L}_{\text{SDS}}$ and $\nabla_{\theta_\tau}\mathcal{L}_{\text{VSD}}$ are stochastic due to Monte Carlo approximation of the expectation. To resolve this, we simply fix the random seed for each iteration, which works well empirically.

Parallelizing Variational Score Distillation. Variational Score Distillation requires training an additional LoRA model to adapt to the distribution of the current generated results. There are two possible solutions for parallelization: (1) we can update LoRA parameters similarly as our 3D parameters, by calculating their gradients on separate GPUs and aggregating them on the remaining GPU; (2) we can keep different LoRA models on different GPUs and separately update each without passing them back for aggregation. The first approach results in more accurate gradients as it seeks to parallelize updates of a single LoRA model, but we find that passing around LoRA parameters across GPUs is very expensive and can undermine any speed gain for the actual 3D model updates. We therefore avoid this approach. The second solution provides less accurate LoRA gradients in theory because our 3D model parameters can be passed to different LoRA models on different GPUs at any iteration, which can provide different LoRA updates at any point in time. However, we observe that since each of our 3D model parameters in the window is randomly allocated to different GPUs for gradient calculation, there is approximately equal probability that each LoRA model will observe all models in the window. This means, roughly speaking, each LoRA model will learn the distribution of all 3D models in the current window. Therefore, updating LoRA separately on different GPUs gives valid guidance and eliminates the need for communicating additional parameters across GPUs.

Adaptive threshold. Fixed-point errors control how close one is to the true trajectory. Smaller thresholds lead to slow convergence and larger thresholds lead to worse generation quality. In practice, we observe the threshold can be different for different prompts and 3D representations. To avoid the need for excessively tuning thresholds, we propose to adaptively update the threshold by the exponential moving average (EMA) of the mean or median error of the current window. With EMA decay rate γ, window of size p starting at time τ and iteration k, and assume the parameter has dimension D, we update threshold e by

$$e \leftarrow \gamma e + (1-\gamma) * M(\{\frac{1}{D} d(\theta_{\tau+i}^{k+1}, \theta_{\tau+i}^k)^2\}_{i=1}^p) \quad (14)$$

where M is a mean or median function. We investigate the effect of the EMA parameters in ablation studies.

Our final method is depicted in Figure 3, and we provide a detailed practical algorithm in Appendix B.

5. Experiments

Running DreamPropeller for T iterations is guaranteed to give the same results as the original method, and it empirically often requires much fewer steps for convergence, a property we empirically investigate in this section. We first show that our method achieves consistent more than 4x speedup when applied to a variety of 3D representations and different score distillation frameworks. We then conduct ablation studies on the proposed practical decisions.

5.1. Accelerating Text-to-3D Generation

We choose baselines that represent the most prominent 3D representations, namely NeRF from DreamFusion [29], DMTet from Magic3D [15], SDF from coarse-stage TextMesh [44] (see Appendix D for detail), and 3D Gaussian Splatting from DreamGaussian [43], and directly apply our wrapper to them. In addition, we show that our algorithm can similarly be adapted to the recently proposed VSD from ProlificDreamer [46]. Our DreamPropeller wrapper is modular and agnostic to the calculation of gradient updates via score distillation, and we test its performance by comparing each of the baselines' runtime and quality with and without our wrapper.

Data and metrics. We use 30 prompts from the DreamFusion gallery to test each algorithm. We run each framework with each individual prompt and record its wallclock runtime in seconds. Following [10, 24, 44], we use CLIP R-Precision [27] for measuring semantic alignment between the generated asset and its input prompt. CLIP R-Precision is the retrieval accuracy of a prompt from the set of prompts given a generated image conditioned on this prompt, and we use top-1 retrieval accuracy for all experiments. We additionally use CLIP FID score [14] compared against the ImageNet 2012 validation set [33] for generation quality.

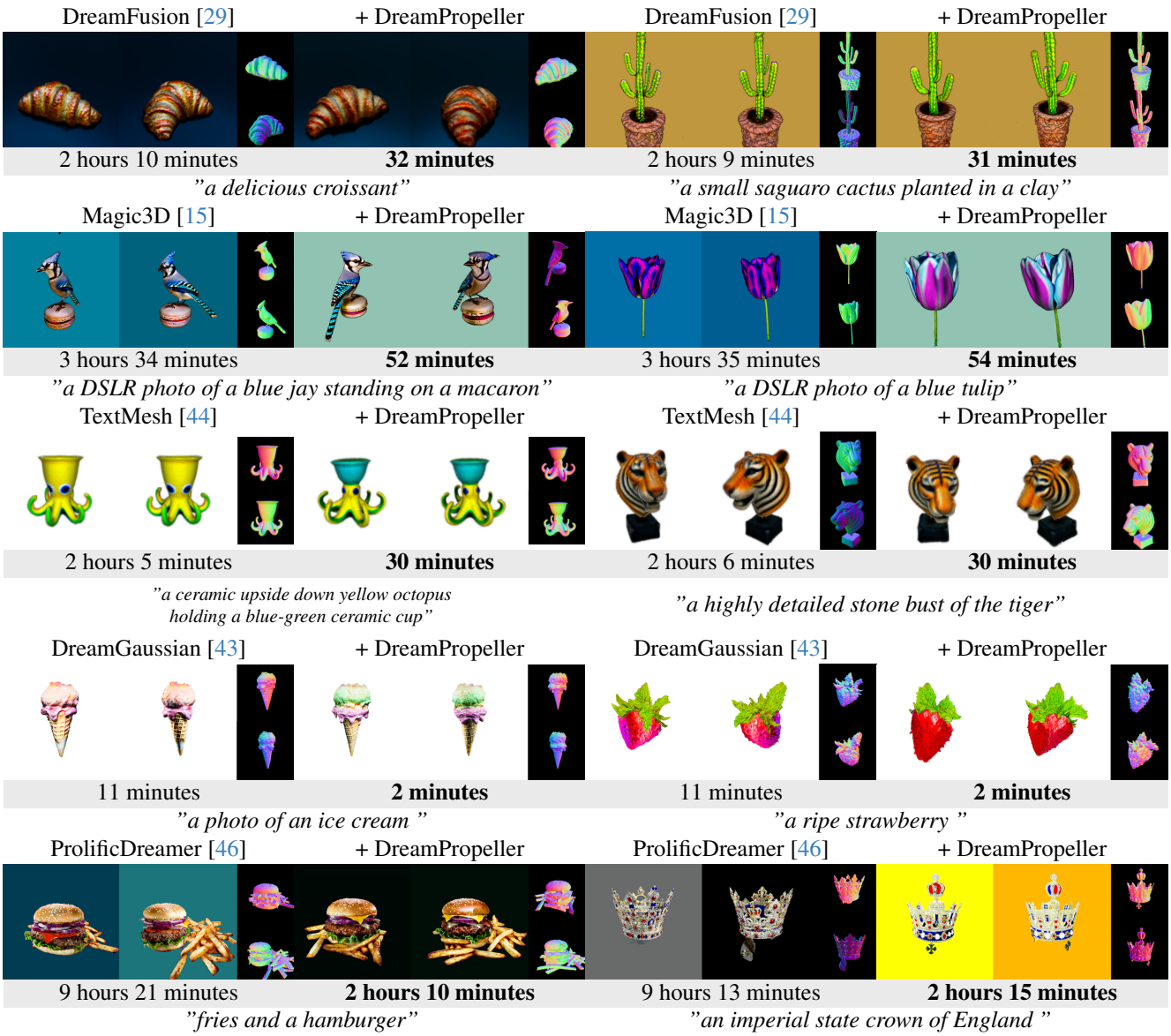
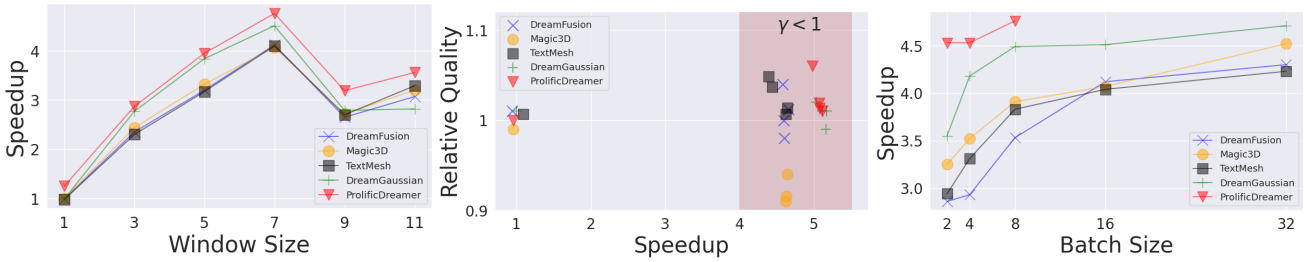


Figure 4. Visual comparisons. Methods using DreamPropeller achieve equally high-quality generation with a much shorter runtime.

Speedup is calculated as the ratio of wall-clock runtime for baseline to our method – higher is better.

Evaluation. As larger batch size can lead to higher-quality generation [4, 15, 26, 46], we use batch size 16 for all baselines but ProlificDreamer, for which, due to memory constraints, we use batch size 8 for the first 5000 steps (rendered at 64×64 resolution) and batch size 2 for the remaining steps (rendered at 512×512 resolution). We use 8 NVIDIA A100 PCIe GPUs for all experiments, which default to window size 7. More details can be found in Appendix D. We evaluate R-Precision and FID_{CLIP} using all rendered images from all generated shapes. Quantitative results can be found in Table 2 and qualitative results are shown in Figure 4.

Notice that with competitive generation quality, we consistently achieve more than 4x speedup for all frameworks and algorithms, with the most speedup for ProlificDreamer. This is in line with the intuition that our parallelization algorithm is more effective when each iteration's GPU workload becomes heavier, which is the case for ProlificDreamer due to it keeping and updating another LoRA model on the fly. The heavy workload is effectively delegated to different GPUs. In theory, the heavier the GPU workload is, the more the optimization benefits from our framework. Also note that DreamGaussian experiences competitive speedup as ProlificDreamer. This is because its original implementa-



(a) Ablation on window size. (b) Ablation on threshold EMA rate. (c) Ablation on batch size.

Figure 5. Ablation studies on practical choices. Speedup is the ratio of baseline wall-clock runtime to our wall-clock runtime. Relative quality is the ratio of baseline FID_{CLIP} to our FID_{CLIP}.

	R-Precision ↑	FID ↓	Runtime (s) ↓	Speedup
DreamFusion [29]	**82.70**	60.91	8080	4.22x
+ DreamPropeller	79.57	**59.88**	**1910**	
Magic3D [15]	**88.45**	59.24	13470	4.17x
+ DreamPropeller	87.18	**59.19**	**3230**	
TextMesh [44]	**88.06**	64.19	7690	4.18x
+ DreamPropeller	84.43	**62.12**	**1830**	
DreamGaussian [43]	83.28	58.33	700	4.67x
+ DreamPropeller	**83.85**	**58.26**	**150**	
ProlificDreamer [46]	94.11	49.66	7390	4.69x
+ DreamPropeller	**96.12**	**49.65**	**3710**	

Table 2. Quantitative evaluation on 30 prompts from the DreamFusion gallery. Runtime is reported in seconds. Our method achieves competitive quality while provide more than 4x speedup.

tion requires sequential rendering of each instance within a batch. This scales its GPU runtime linearly with batch size, so our framework gives better speedup. We also notice that our results do not exactly match those of baselines. This is likely due to the fixed-point error not being low enough for converging to the fixed-point solution, and Adam takes the parameters to slightly different but equally valid local optima through momentum-based gradient updates.

5.2. Ablation Studies

We further conduct ablation studies for the practical choices to investigate their effectiveness.

Effect of window size. By default, we set the window size to be 1 less than the number of GPUs, since we need the remaining one for sequential rollout. Allowing all other GPUs to do independent work intuitively maximizes speedup. However, how does the speedup scale when window size is independent of the number of GPUs? For our experiment in Figure 5a, we assume access to 8 NVIDIA A100 PCIe GPUs and adjust the window size from 1 to beyond 7. The speedup of our framework peaks at window size 7, because we fully utilize all GPU resources for maximum parallelization. We also observe a drop in speedup for window size beyond 7. We hypothesize that this is due to longer sliding windows not advancing far enough under GPU resource constraints while requiring more FLOPs, thus causing slower speedup.

Effect of threshold adaptivity. The adaptivity parameter aims to stabilize speedup while maintaining generation quality. We show in Figure 5b its ablation study. Relative quality is calculated as the ratio of baseline FID_{CLIP} to our FID_{CLIP} – a higher ratio means better relative quality. For each model, 5 settings are tested with $\gamma \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ and initial thresholds set to 1% of the default thresholds. This is to maximally bottleneck our framework to test how much γ counteracts an ill-initialized threshold. We observe a large gap between results from $\gamma = 1$ and $\gamma < 1$ (in red region), implying as long as $\gamma < 1$ (*i.e.* our framework is adaptive), all models achieve similar speedup without significantly degrading quality.

Effect of batch size. As SDS/VSD requires evaluating expectation over diffusion time t, one is motivated to use a large batch size of t for more accurate estimate of the guidance score, which can lead to higher-quality generation [4, 15, 26, 46]. As such, this increases computational demand for GPU per iteration, a regime where our method is particularly beneficial. As shown Figure 5c, the speedup of our method scales with increasing batch sizes because of the increase in computational intensity per iteration. In particular, our method benefits VSD more even when batch size is small because the LoRA training is costly and is effectively amortized across GPUs because their parameters are never communicated across processes.

6. Conclusion

In this work, we introduce DreamPropeller, a drop-in acceleration framework for all existing score distillation-based text-to-3D generation methods. DreamPropeller can achieve more than 4x speedup for various predominant 3D representations and benefits more from heavier demand for GPU computation per iteration. We hope our framework serves as a significant step towards usable high-quality text-to-3D methods and inspires more advancements to come.

References

[1] Monther Alfuraidan and Qamrul Ansari. *Fixed Point Theory and Graph Theory: Foundations and Integrative Approaches*. Academic Press, 2016. 2

[2] Haotian Bai, Yuanhuiyi Lyu, Lutao Jiang, Sijia Li, Haonan Lu, Xiaodong Lin, and Lin Wang. CompoNeRF: Text-guided multi-object compositional NeRF with editable 3D scene layout. 2023. 2

[3] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Qinsheng Zhang, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, Tero Karras, and Ming-Yu Liu. EDiff-I: Text-to-image diffusion models with an ensemble of expert denoisers. 2022. 2

[4] Rui Chen, Yongwei Chen, Ningxin Jiao, and Kui Jia. Fantasia3D: Disentangling geometry and appearance for high-quality Text-to-3D content creation. 2023. 2, 7, 8

[5] Zilong Chen, Feng Wang, and Huaping Liu. Text-to-3D using gaussian splatting. 2023. 2, 3, 4

[6] Dana Cohen-Bar, Elad Richardson, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Set-the-Scene: Global-Local training for generating controllable NeRF scenes. 2023. 2

[7] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing Text-to-Image generation using textual inversion. 2022. 2

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Adv. Neural Inf. Process. Syst.*, 27, 2014. 2

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Adv. Neural Inf. Process. Syst.*, 33:6840–6851, 2020. 1, 2

[10] Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 867–876. openaccess.thecvf.com, 2022. 6

[11] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D gaussian splatting for Real-Time radiance field rendering. *ACM Trans. Graph.*, 42(4):1–14, 2023. 2, 3, 4

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. 4

[13] Nupur Kumari, Bingliang Zhang, Richard Zhang, Eli Shechtman, and Jun-Yan Zhu. Multi-concept customization of text-to-image diffusion. pages 1931–1941, 2022. 2

[14] Tuomas Kynkäänniemi, Tero Karras, Miika Aittala, Timo Aila, and Jaakko Lehtinen. The role of ImageNet classes in fréchet inception distance. 2022. 6

[15] 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. 2022. 2, 6, 7, 8

[16] Minghua Liu, Chao Xu, Haian Jin, Linghao Chen, Varma T Mukund, Zexiang Xu, and Hao Su. One-2-3-45: Any single image to 3D mesh in 45 seconds without Per-Shape optimization. 2023. 2, 3

[17] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9298–9309. openaccess.thecvf.com, 2023. 2, 3, 4

[18] Ying-Tian Liu, Yuan-Chen Guo, Vikram Voleti, Ruizhi Shao, Chia-Hao Chen, Guan Luo, Zixin Zou, Chen Wang, Christian Laforte, Yan-Pei Cao, and Others. threestudio: a modular framework for diffusion-guided 3D generation. *cg.cs.tsinghua.edu.cn*. 1

[19] Zhiheng Liu, Ruili Feng, Kai Zhu, Yifei Zhang, Kecheng Zheng, Yu Liu, Deli Zhao, Jingren Zhou, and Yang Cao. Cones: Concept neurons in diffusion models for customized generation. 2023. 2

[20] Jonathan Lorraine, Kevin Xie, Xiaohui Zeng, Chen-Hsuan Lin, Towaki Takikawa, Nicholas Sharp, Tsung-Yi Lin, Ming-Yu Liu, Sanja Fidler, and James Lucas. ATT3D: Amortized Text-to-3D object synthesis. 2023. 2

[21] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3D point cloud generation. pages 2837–2845, 2021. 1

[22] 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. 2021. 1, 2

[23] 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. *Commun. ACM*, 65(1):99–106, 2021. 1, 2, 3, 4

[24] Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. CLIP-Mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 Conference Papers*, number Article 25 in SA '22, pages 1–8, New York, NY, USA, 2022. Association for Computing Machinery. 6

[25] Ron Mokady, Amir Hertz, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Null-text inversion for editing real images using guided diffusion models. pages 6038–6047, 2022. 2

[26] Zijie Pan, Jiachen Lu, Xiatian Zhu, and Li Zhang. Enhancing High-Resolution 3D generation through pixel-wise gradient clipping. 2023. 7, 8

[27] Dong Huk Park, Samaneh Azadi, Xihui Liu, Trevor Darrell, and Anna Rohrbach. Benchmark for compositional Text-to-Image synthesis. 2021. 6

[28] Ryan Po and Gordon Wetzstein. Compositional 3D scene generation using locally conditioned diffusion. 2023. 2

[29] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. 2022. 1, 2, 3, 6, 7, 8

[30] Guocheng Qian, Jinjie Mai, Abdullah Hamdi, Jian Ren, Aliaksandr Siarohin, Bing Li, Hsin-Ying Lee, Ivan Skorokhodov, Peter Wonka, Sergey Tulyakov, and Bernard Ghanem. Magic123: One image to High-Quality 3D object generation using both 2D and 3D diffusion priors. 2023. 2, 3

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

the IEEE/CVF conference on computer vision and pattern recognition, pages 10684–10695. openaccess.thecvf.com, 2022. 1, 2

[32] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. DreamBooth: Fine tuning text-to-image diffusion models for subject-driven generation. pages 22500–22510, 2022. 2

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. 2014. 6

[34] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, and Others. Photorealistic text-to-image diffusion models with deep language understanding. *Adv. Neural Inf. Process. Syst.*, 35:36479–36494, 2022. 2

[35] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5B: An open large-scale dataset for training next generation image-text models. pages 25278–25294, 2022. 1, 2

[36] Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. 2023. 2, 3, 6

[37] J Ryan Shue, Eric Ryan Chan, Ryan Po, Zachary Ankner, Jiajun Wu, and Gordon Wetzstein. 3D neural field generation using triplane diffusion. pages 20875–20886, 2022. 1

[38] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2256–2265, Lille, France, 2015. PMLR. 1, 2

[39] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[40] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based generative modeling through stochastic differential equations. 2020. 1, 2

[41] Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. Accelerating feedforward computation via parallel nonlinear equation solving. In *International Conference on Machine Learning*, pages 9791–9800. PMLR, 2021. 2

[42] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018. 2

[43] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. DREAMGAUSSIAN: GENERATIVE GAUSSIAN SPLAT- TING FOR EFFICIENT 3D CONTENT CREATION. 1, 2, 3, 4, 5, 6, 7, 8

[44] Christina Tsalicoglou, Fabian Manhardt, Alessio Tonioni, Michael Niemeyer, and Federico Tombari. TextMesh: Generation of realistic 3D meshes from text prompts. 2023. 2, 6, 7, 8

[45] Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A Yeh, and Greg Shakhnarovich. Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12619–12629. openaccess.thecvf.com, 2023. 1, 2, 3

[46] 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. 2023. 1, 2, 3, 6, 7, 8

[47] Haohan Weng, Tianyu Yang, Jianan Wang, Yu Li, Tong Zhang, C L Philip Chen, and Lei Zhang. Consistent123: Improve consistency for one image to 3D object synthesis. 2023. 2

[48] Taoran Yi, Jiemin Fang, Guanjun Wu, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Qi Tian, and Xinggang Wang. GaussianDreamer: Fast generation from text to 3D gaussian splatting with point cloud priors. 2023. 2, 3, 4

[49] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. pages 3836–3847, 2023. 1, 2

[50] Linqi Zhou, Yilun Du, and Jiajun Wu. 3D shape generation and completion through point-voxel diffusion. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5826–5835. IEEE, 2021. 1