SUPPLEMENTARY MATERIALS

A EXPERIMENT DETAILS

A.1 IMPLEMENTATION DETAILS

To clarify the difference between with/without negative prompts, and with/without Perp-Neg, we provide an illustrative depiction in Figure 8. Our Perp-Neg does not require additional training or fine-tuning, and is implemented in the sampling pipeline. A detailed implementation in each timestep is shown in Algorithm 1.

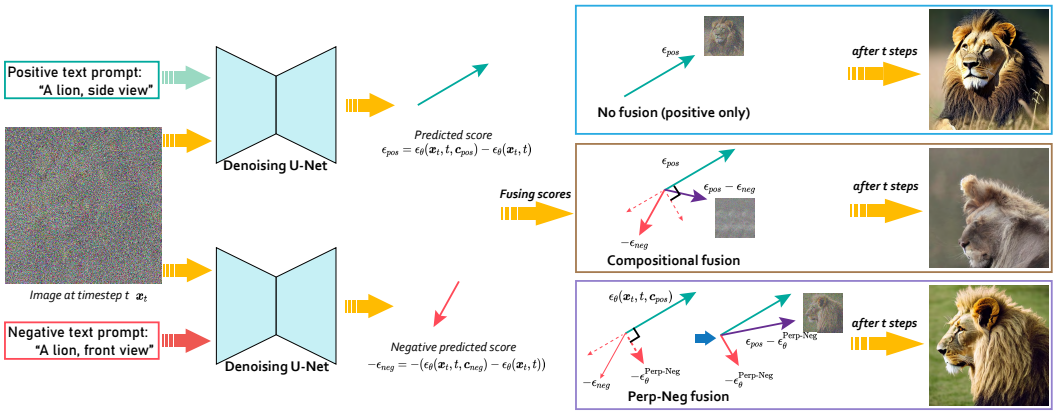


Figure 8: Illustrative depiction of Perp-Neg, and comparison with sampling without the usage of negative prompt and compositional negative prompt fusion.

Algorithm 1 Perp-Neg Pseudocode at timestep t, PyTorch-like style

```
# epsilon: diffusion model
# x: the noisy image at the current timestep
# t: the current time step t
# c: main text prompt
# c_neg: auxiliary negative text prompt list
# a: classifier-free guidance scale
# w_pos: weight of main text prompts
# w_neg: weight list of negative text prompts

def Perp_neg(M, x, t, c, c_neg, a, w_pos, w_neg):
   # get the main component
   e_main = epsilon(x, t, c) - epsilon(x, t)

   # proceed with auxiliary negative text prompts
   for i, text_negative in enumerate(c_neg):
      e_i = epsilon(x, t, text_negative) - epsilon(x, t)
      accum_grad -= w_neg[i] * get_prependicualr_component(e_i, e_main) # accumulate the
           negative gradients in the opposite direction

   e = epsilon(x, t) + a * (w_pos * z_main + accum_grad) # update the noisy image at the
        current timestep to the next step (eq. 8)

   return e # return the final prediction at time step t.


# definition of "get_prependicualr_component"
def get_prependicualr_component(x, y):
   # x: gradient of principle component
   # y: gradient of auxiliary component
   proj_x = ((torch.mul(x, y).sum())/(torch.norm(y)**2)) * y # cosine projection of x on
         y
   return x - proj_x # get perpendicular vector of x
```

For 2D generation, we implement our Perp-Neg into Stable Diffusion v1.4 pipeline. We adopt 50 DDIM steps, and fix the guidance scale as $a = 7.5$ and the positive weight $w_{\text{pos}} = 1$, for each generation. For negative weight, the value may vary and we find generally in the range [-5, -0.5] can produce satisfactory images. In the 2D view generation experiments, we fix the negative weights to

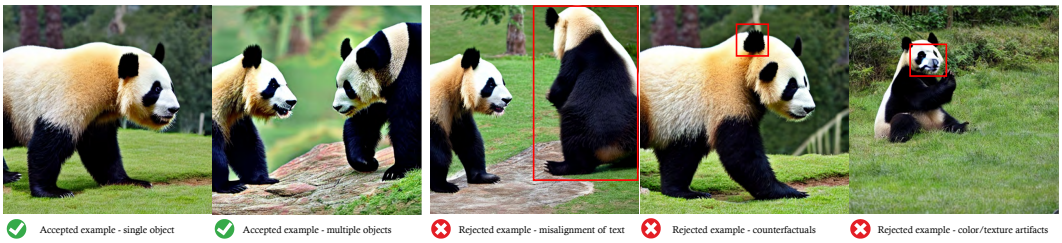


Figure 9: Example of the accepted/rejected generation.

$w_{\text{neg}} = -1.5$ when there is only one negative prompt, and set $w_{\text{neg}_1} = w_{\text{neg}_2} = -1$ when there are two negative prompts. The results are generated across seeds 0-49. For the interpolation between two views, we normally interpolate r_{inter} with stride 0.25 between 0 and 1 to have 5 images in total. Moreover, for 3D generation, we employ Perp-Neg into Stable Dreamfusion. The results of our baselines are reproduced with their open-source code[1][2]. All experiments are conducted using Pytorch 1.10 on a single NVIDIA-A5000 GPU.

A.2 Details About 2D Experiment

For the choice of prompt in the 2D experiment, we aim to test two groups of text prompts that generate the side view and the back view of the objects, which are considered simple and complex cases, respectively. For each group, when using the negative prompts, we use the complementary view or the combination of the other two views, *e.g.*, in the case of using the "side" view in the positive prompt, we use the "front" view in the negative prompt. Both positive and negative prompts follow the basic prompt pattern but respectively adopt positive and negative weight in the fusing stage.

Criteria for successful view generation count Here we elaborate on the criteria for the successful generation count in our quantitative experiments in Section 4.1. We reject the image samples with the following criteria and examples are shown in Figure 9:

- The images that do not show requested object(s) or view. Note that if the generated image contains multiple objects, and one of them is not positioned in the correct view, the image will still be rejected.
- The images show hallucination including counterfactual details, for example, a panda has three ears.
- The images have color or texture artifacts that make the images not realistic.

More details: On the side view and back view generation, in each group, we adopt 3 combinations of the complementary view into the negative prompts, *e.g.*, for the case that side view is used in the positive prompt, we use front view, back view and both front and back view in the negative prompt.

A.3 Further Extensions

Although we only provide an application of Perp-Neg using SDS loss, we will investigate its application in novel view synthesis Deng et al. (2022); Liu et al. (2023); Tang et al. (2023); Vinod et al. (2023); Watson et al. (2022), conditional 3D generation Karnewar et al. (2023); Nichol et al. (2022); Po & Wetzstein (2023); Raj et al. (2023), editing Ceylan et al. (2023); Haque et al. (2023); Mikaeili et al. (2023); Voynov et al. (2023) and adding texture Richardson et al. (2023).

B 3D Experiment Details and Results:

To determine the negative prompt weight functions f, we used the general form of a shifted exponential decay of the form $f_{\cdot}(r) = a \exp(-b * r) + c$, where a, b, and c are greater than or equal to

[1]https://github.com/energy-based-model/Compositional-Visual-Generation-with-Composable-Diffusion-Models-PyTorch

[2]https://github.com/ashawkey/stable-dreamfusion

zero. We set the parameters of the f functions separately for each text prompt by generating 2D interpolation samples for 10 different random seeds and then selecting the parameters with the highest accuracy in following the conditioned view. We observed that parameters that allow better interpolation in 2D cases directly relate to the parameters that better help with the Janus problem in 3D. We also noted that if the interpolation between two views remained unchanged while varying the angle for a large range, then the 3D-generated scene was more likely to have a flat geometry from multiple views, resulting in the Janus problem. To overcome this, we perturbed the interpolation factor r with random noise, calculated the interpolated text embedding and their related negative weights, and thus, the model is less likely to generate identical photos from a range of views. We provided the code in the supplementary file for the exact details.

To evaluate the effectiveness of the Perp-Neg in alleviating the Janus problem, we conducted our experiments using prompts that did not depict circular objects. For each prompt, we utilized the DreamFusion method with and without the Perp-Neg algorithm. For Stable-Diffusion DreamFusion, we run 14 trials for each approach with different seeds. Our results indicate the number of successful outputs generated without a Janus problem when using the Perp-Neg algorithm: "a corgi standing" 2 times, "a westie" 5 times, "a lion" 1 time, "a Lamborghini" 5 times, "a cute pig" 0 times, and "Super Mario" 4 times. In contrast, when we ran the model without the Perp-Neg, it failed to generate any correct output except for "a Lamborghini" 4 times and "Super Mario" 2 times. These findings clearly demonstrate the advantages of utilizing the Perp-Neg in mitigating the Janus problem. Please refer to Figure 20 for some of these examples.

For DeepFloyd-IF DreamFusion, we run 6 trials for each approach with different seeds. Our results indicate the number of successful outputs generated without a Janus problem when using the Perp-Neg algorithm: "a Shiba dog wearing sunglasses" 4 times "a marble lion bust" 4 times, "a tiger cub" 3 times, "a corgi taking a selfie" 3 times, "an ancient golden pig" 1 time, and "a DSLR photo of a lion bust" 3 times. In contrast, when we ran the model without the Perp-Neg, it failed to generate any correct output except for "a corgi taking a selfie" 1 time.

We also incorporated Perp-Neg during the high-quality mesh optimization stage of Magic3D. When Perp-Neg was not used, we observed that in a few cases, even if the initial mesh (the output of DreamFusion) did not exhibit the Janus problem, the second stage of Magic3D optimization resulted in a multiface issue with the mesh. However, with the use of Perp-Neg, we never encountered a second-stage multiface issue.

C Additional Experiments

C.1 Case-by-case study

We further conduct case-by-case studies for the previous experiments, where we collect statistics of every tested object per view. We report the averaged acceptance rate across all possible positive and negative combinations in Table 1.

From the table, we find back view is consistently more difficult to generate than the side view. A possible explanation is that the generator may have more information about the front view from the training datasets, and the side view possesses more connection to the front view, while it requires additional knowledge to generate the corresponding back view. In terms of the objects, we find it is less likely to generate faithful images of peacock(s) in the side view, as well as peacock(s) in the back view lion in the back view, as there may have less corresponding training images in the original training datasets.

View	Side			Back		
Object	Lion	Panda	Peacock	Lion	Panda	Peacock
SD	58.0%	44.0%	24.0%	8.0%	28.0%	8.0%
CEBM	14.0%	13.3%	10.7%	0%	4.0%	2.0%
Ours	**80.7%**	**83.3%**	**55.3%**	**49.3%**	**34.0%**	**38.0%**

Table 1: Case-by-case percentage of successful view generations different objects.

C.2 ABLATION STUDY

We also provide ablation studies on the effects of negative prompt weights to see how the weight affects the usage of negative attribute elimination. We show the results of generations using different negative prompt weights w_i in Figure 10-12. We can observe for CEBM, the results are consistently not relevant to the requested text content, no matter the negative prompt weights are small or big. For Perp-Neg, we can see with larger weights, *e.g.*, $w = -0.1$, the generated results are not positioned in the side view. As we decrease the weight, the generated image becomes more relevant to the text. This observation indicates Perp-Neg has better controllability in eliminating the negative attributes in the prompts.

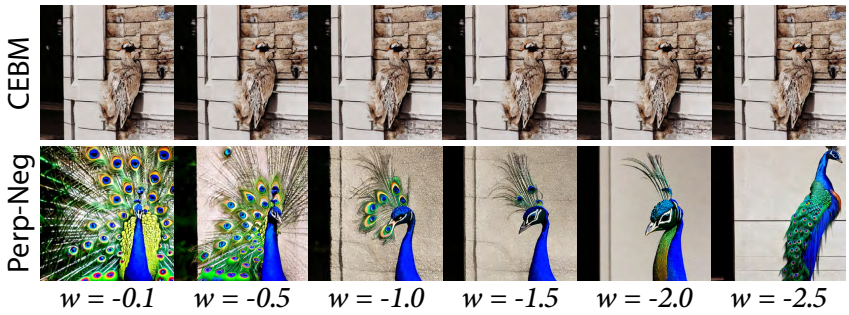


Figure 10: Ablation studies with different negative prompt weights in the generation, side view of peacocks.

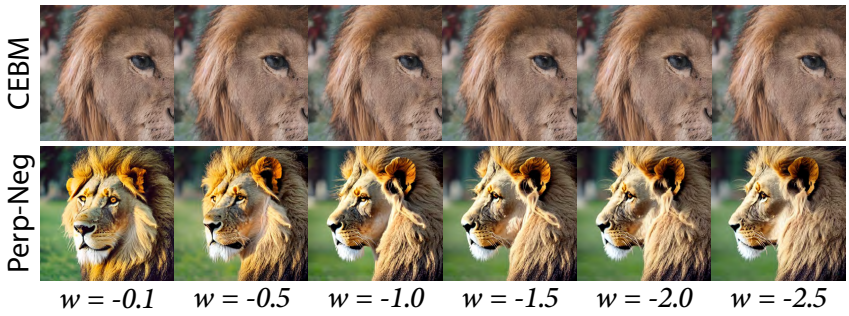


Figure 11: Analogous visualization of lions to Figure 10.

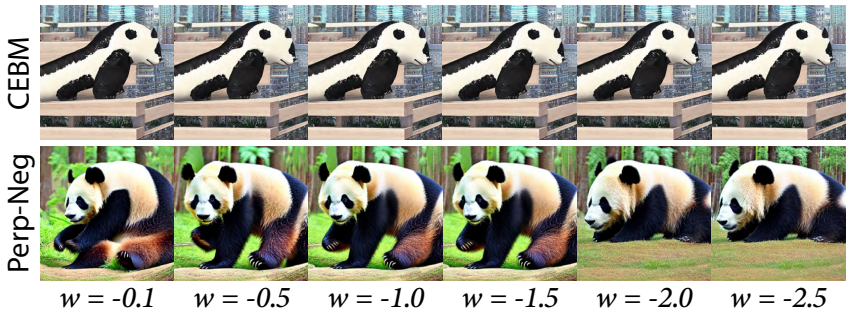


Figure 12: Analogous visualization of pandas to Figure 10.

C.3 ADDITIONAL RESULTS

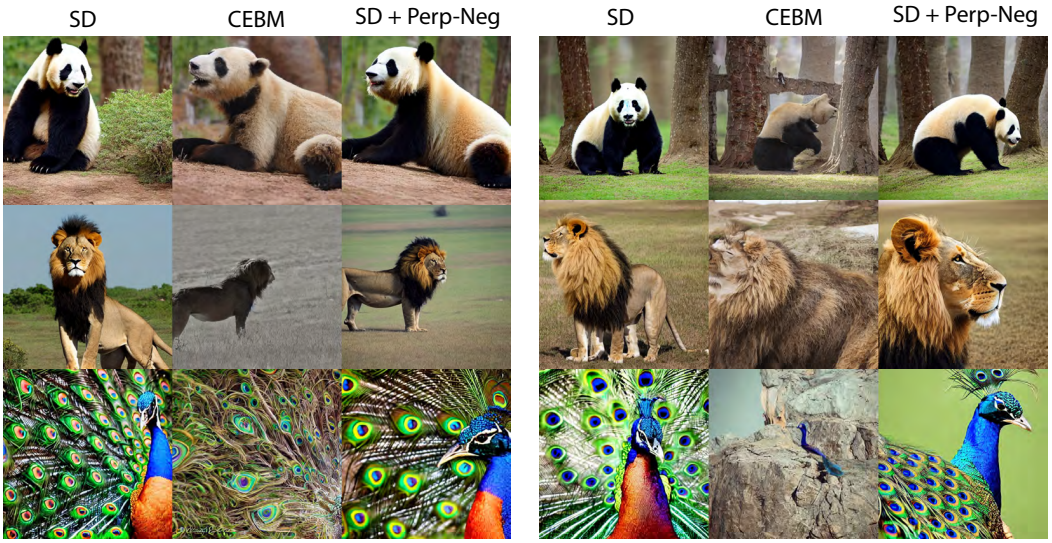


Figure 13: Analogous visualization to Figure 6, side-view generation.

Figure 14: Additional visualization of panda back-view from our experiment. Here we provide a part of those generations with seeds 0-49 using Perp-Neg, including both successful and failed samples. Most of the generations show the semantics of "panda back view".

Figure 15: Analogous visualization to Figure 14 visualization of peacock back-view.

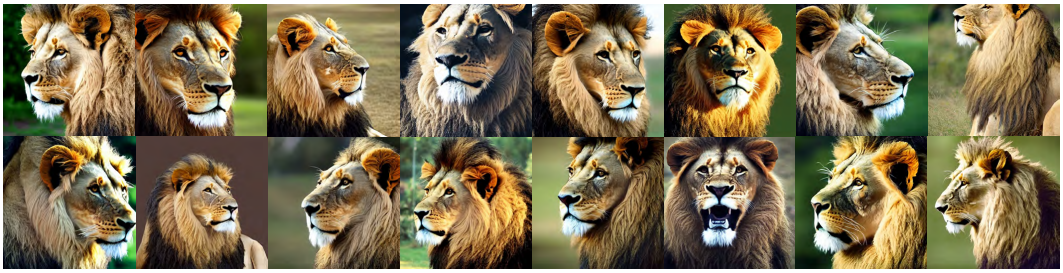


Figure 16: Analogous visualization to Figure 14 visualization of lion side-view.

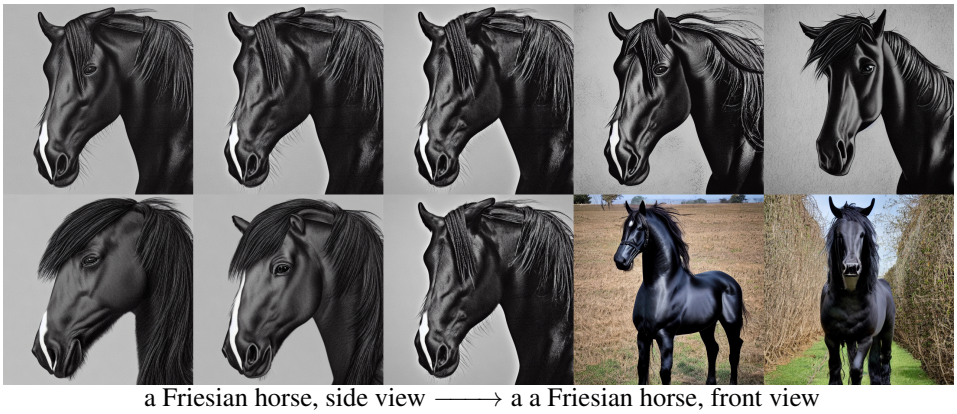
Figure 17: Analogous visualization to Figure 14 visualization of panda side-view.

Figure 18: Analogous visualization to Figure 14 visualization of peacock side-view.

C.4 VIEW INTERPOLATION

In the following, we provide additional qualitative results of 2D generation and view interpolation. And for 3D generation results, please refer to the provided video in the supplementary file. We explore how Perp-Neg can be employed for effective interpolation between two views of an object in 2D as it is needed for 3D cases, as illustrated in Figure **??**.

a giant panda, side view ⟶ a giant panda, front view



a Friesian horse, side view ⟶ a a Friesian horse, front view

a cute tiger cub, side view ⟶ a cute tiger cub, back view

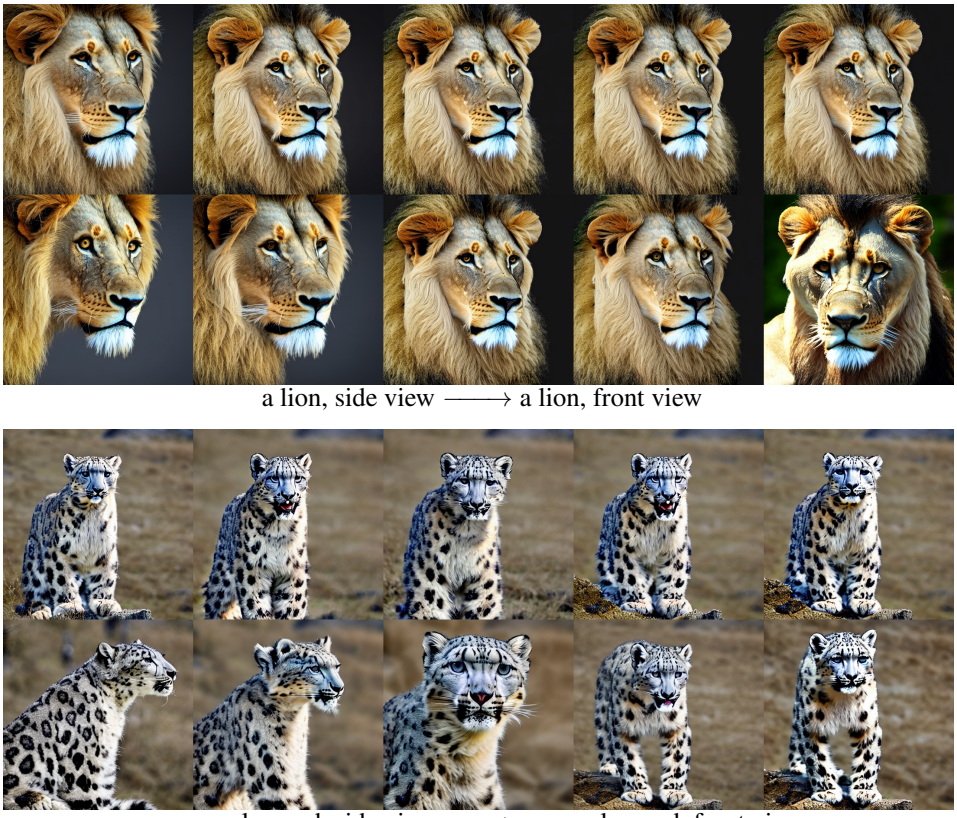


Figure 19: **Qualitative comparison** of view interpolation with/without Perp-Neg. We fixed the seed across different images of each prompt. For each prompt, the top row shows the result of text-embedding interpolation without Perp-Neg. And, the bottom row shows the result of Perp-Neg.

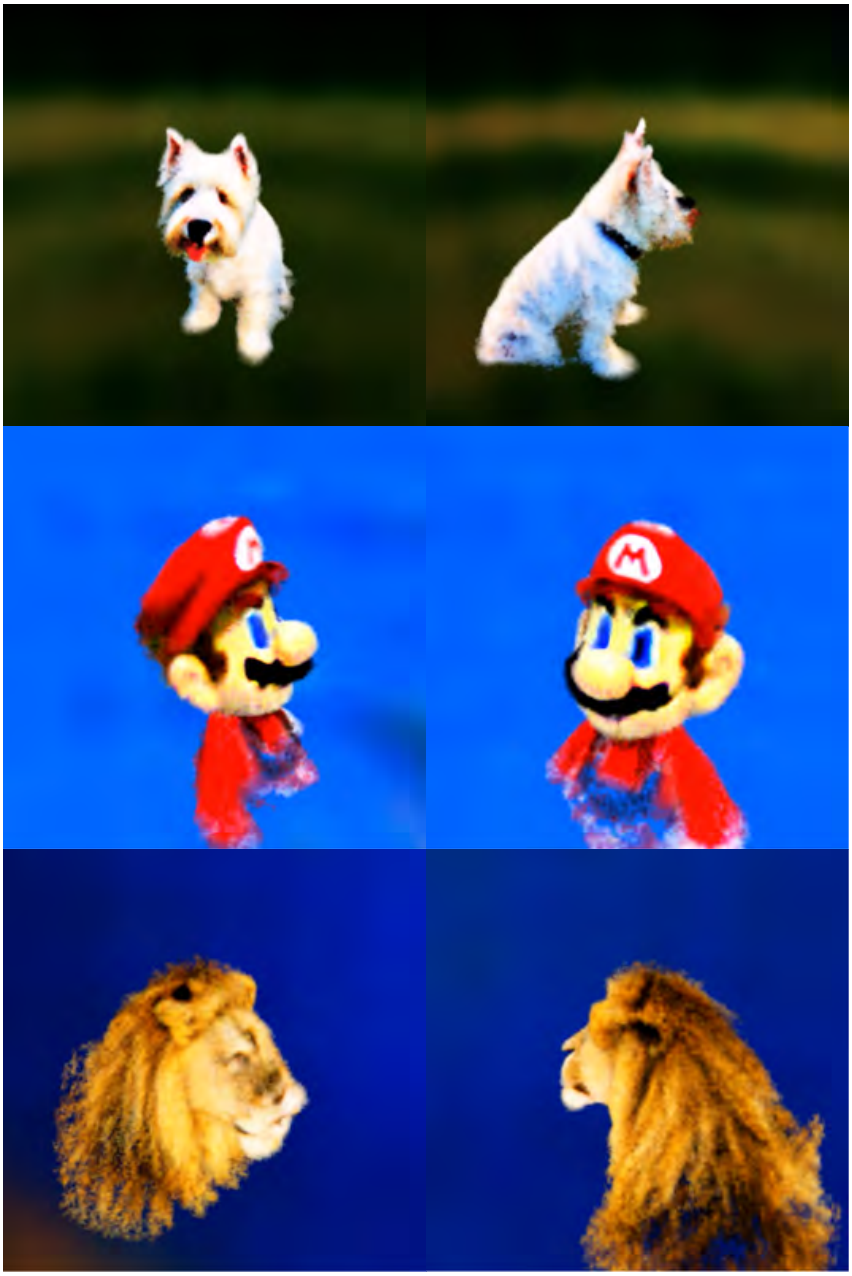


Figure 20: Qualitative examples of Stable-Dreamfusion with Perp-Neg, using prompts “a westie”, “Super Mario” and “a lion.”