Go-with-the-Flow: Motion-Controllable Video Diffusion Models Using Real-Time Warped Noise

Supplementary Material

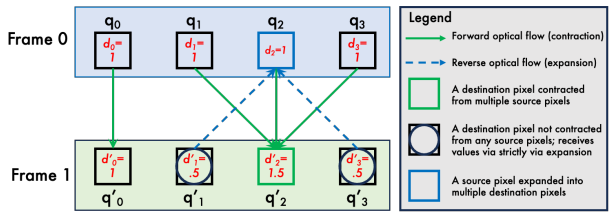


Figure 8. Diagram of our noise warping algorithm. A case example of our algorithm illustrates both the *expansion* and *contraction* cases, along with example density values. Each node represents some noise pixel 'q'. Noise values $q_{0..3}$ are transferred from frame 0 to frame 1 using forward optical flow, and the remaining pixels in frame 1 that did not receive any values obtain their values from frame 0 using reverse optical flow (the *expansion* case). In the *contraction* cases such as q'_2, their densities become the sum of their sources. And in the *expansion* case, where one source pixel spreads out into multiple target pixels, such as q_2 spreading out into q'_1 and q'_3, its density is dispersed.

5. Method

5.1. Noise Warping Algorithm Details

Continuing the discussion in Sec. 3, we elaborate on our noise warping algorithm. We provide algorithmic pseudocode for our noise-warping algorithm in Algorithm 1, in pytorch-like pseudocode in Fig. 17. The full source code and models are open source and can be found on GitHub. We illustrate the contraction and expansion processes of our noise warping algorithm in Fig. 8.

5.2. Gaussianity preservation of our noise warping algorithm

In this section, we discuss our noise warping algorithm, providing a formal proof of its Gaussianity preservation properties. We also present an illustrative example that demonstrates how noise that undergoes expansion and subsequent contraction returns to its original state, showcasing how our noise warping algorithm maintains the underlying Gaussian distribution throughout the warping process.

Proof. For each $(x, y) \in V$, $R(x, y)$ is a collection of up-

Algorithm 1 Go-with-the-Flow next-frame warping

Input: previous-frame noise $q \in \mathbb{R}^D$, previous-frame density $p \in \mathbb{R}^D$, forward flow $f : D \to \mathbb{N}^2$, backward flow $f' : D \to \mathbb{N}^2$.
Let $G = (V, V', E)$ be a bipartite graph with $V = D$, $V' = D$ and edge set $E = \{\}$ to be constructed.
for v in V **do** ▷ Contraction
  $E \leftarrow E \cup (v, v + f(v))$ if $v + f(v') \in D$
end for
for v' in V' **do** ▷ Expansion
  if $\deg_G(v') = 0$ **then** ▷ $\deg_G(v)$ denote the degree of v in G
    $E \leftarrow E \cup (v' + f'(v'), v')$ if $v' + f'(v') \in D$
  end if
end for
for v in V **do** ▷ Conditional white noise sampling
  $d \leftarrow \deg_G(v)$
  Sample $Z \sim \mathcal{N}(0, I_d)$, and set $S \leftarrow \sum_{i=1}^{d} Z_i$
  $X_i \leftarrow \frac{q(v)}{d} + \frac{1}{\sqrt{d}}(Z_i - \frac{S}{d})$ for $i \in [d]$
  $R(v) \leftarrow \{X_i\}_{i \in [d]}$
end for
for (v') in V' **do** ▷ Compute next-frame noise and density
  $q'(v') \leftarrow 0, p'(v') \leftarrow 0, s \leftarrow 0$
  for v in V such that $(v, v') \in E$ **do**
    $d \leftarrow \deg_G(v), \alpha \leftarrow \frac{p(v)}{d}$
    $q'(v') \leftarrow q'(v') + \alpha R(v).\text{pop}()$
    $p'(v') \leftarrow p'(v') + \alpha$
    $s \leftarrow s + \alpha^2 \frac{1}{d}$
  end for
  if $s = 0$ **then**
    Sample $q'(v') \sim \mathcal{N}(0, 1)$
  else
    $q'(v') \leftarrow \frac{q'(v')}{\sqrt{s}}$ ▷ Renormalize to unit variance
  end if
end for
return next-frame noise and density q', p'.

sampled noise X_i, where

$$\mathbb{E}[X_i] = \mathbb{E}[\frac{q(x,y)}{d}] + \mathbb{E}[\frac{1}{\sqrt{d}}(Z_i - \frac{S}{d})] = 0$$

$$\begin{aligned}\text{Var}(X_i) &= \text{Var}(\frac{q(x,y)}{d}) + \text{Var}(\frac{1}{\sqrt{d}}(Z_i - \frac{S}{d})) \\ &= \frac{1}{d^2} + \frac{1}{d}\text{Var}(\frac{d-1}{d}Z_i - \sum_{j \neq i}\frac{Z_j}{d}) \\ &= \frac{1}{d^2} + \frac{1}{d}\frac{(d-1)^2 + (d-1)}{d^2} = \frac{1}{d},\end{aligned}$$

where we used the fact that $q(x, y)$ and Z_i's are i.i.d. standard Gaussians. Since X_i is constructed as a weighted sum of Gaussians, itself is also a Gaussian. Moreover, for $i \neq j$, we compute

$$\begin{aligned}
&\text{Cov}(X_i, X_j) \\
=&\text{Cov}(\frac{q(x,y)}{d} + \frac{1}{\sqrt{d}}(Z_i - \frac{S}{d}), \frac{q(x,y)}{d} + \frac{1}{\sqrt{d}}(Z_j - \frac{S}{d})) \\
=&\frac{1}{d^2} + \frac{1}{d}\mathbb{E}[(Z_i - \frac{S}{d})(Z_j - \frac{S}{d})] \\
=&\frac{1}{d^2} + \frac{1}{d}(0 - 2\frac{\mathbb{E}[Z_i S]}{d} + \frac{\mathbb{E}[S^2]}{d^2}) \\
=&\frac{1}{d^2} + \frac{1}{d}(-\frac{2}{d} + \frac{1}{d}) = 0.
\end{aligned}$$

Hence all X_i's are independent.

For each $(x', y') \in V'$, if $\deg_G((x', y')) = 0$, then $q'(x', y')$ is sampled as an independent standard Gaussian. Otherwise, the output noise pixel $q'(x', y')$ is built as a weighted sum of $R(x, y).\text{pop}()$ for each edge $((x, y), (x', y')) \in E$, where $R(x, y).\text{pop}()$ is an independent Gaussian of mean 0 and variance $\frac{1}{\deg_G((x,y))}$. Hence $q'(x', y')$ is also a Gaussian with mean 0. The variable s after executing the inner for loop thus represents the variance of $q'(x', y')$, so the renormalization at the end brings $q'(x', y')$ back to a standard Gaussian. Since the composing X_i's are independent, the resulting noise q' should also have an independent Gaussian in each pixel. □

Example 1 (Exact recovery of *expansion-contraction*)**.** *Consider the following evolution of noise across three frames with forward flows $f_{i\to j}$ going from frame i to frame j with $i + 1 = j$ (and backward flow if $i - 1 = j$). Suppose at frame 1, a pixel $v \in D$ with density 1 has noise q. Suppose further that v'_a is a pixel at frame 2 such that $f^{-1}_{1\to 2}(v'_a) = \{v\}$, and $v'_b \in D$ is the only pixel at frame 2 such that $f^{-1}_{1\to 2}(v'_b) = \varnothing$ and $f_{2\to 1}(v'_b) = v$. This represents the scenario where v is expanded into two pixels v'_a, v'_b. Then Algorithm 1 with forward flow $f_{1\to 2}$ and backward flow $f_{2\to 1}$ will result in v'_a having density $1/2$ and noise $\frac{q}{2} + \frac{1}{\sqrt{2}}(\frac{Z_a - Z_b}{2})$, and v'_b having density $1/2$ and noise $\frac{q}{2} + \frac{1}{\sqrt{2}}(\frac{Z_b - Z_a}{2})$, where Z_a and Z_b are i.i.d. standard Gaussians. Now, from frame 2 to frame 3, suppose there exists a pixel v'' such that $f^{-1}_{2\to 3}(v'') = \{v'_a, v'_b\}$, i.e., they both v'_a and v'_b contract to v'', and that $f_{3\to 2}(D) \cap \{v'_a, v'_b\} = \varnothing$. Then Algorithm 1 with forward flow $f_{2\to 3}$ and backward flow $f_{3\to 2}$ will result in v'' having density 1 and noise q, hence deterministically recovering the noise and density of v in frame 0.*

5.3. Implementation details

We fine-tune the recent state-of-the-art open-source video diffusion model, CogVideoX-5B [60], on both its T2V and I2V variants. We use a large general-purpose video dataset composed of 4M videos with resolution ≥720×480 ranging from approximately 10 to 120 seconds in length, with paired texts captioned by CogVLM2 [52]. We used 8 NVIDIA A100 80GB GPUs over the course of 40 GPU days, for 30,000 iterations using a rank-2048 LoRA [22] with a learning rate of 10^{-5} and a batch size of 8.

Our method is data-agnostic and model-agnostic. It can be used to add motion control to arbitrary video diffusion models while only processing the noise sampling during fine-tuning. For example, it also works with AnimateDiff [15] fine-tuned with the WebVid dataset [2], trained on 8×40GB A100 GPUs over a period of 2 days with 12 frames and 256 × 320 resolution. See its qualitative results in Fig. 16 in the supplementary material.

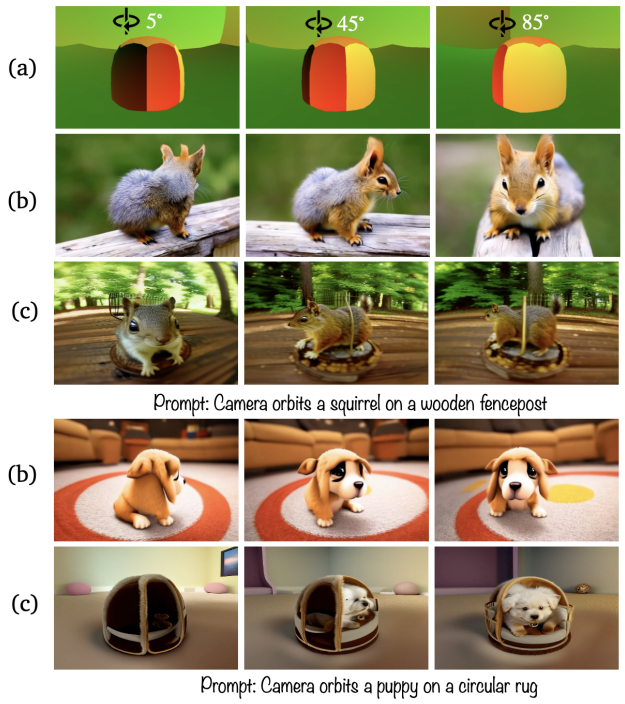


Figure 9. Qualitative comparisons of camera movement video generation of our method (b) and MotionClone (c) using a turning source video (a).

6. Additional Results

In Fig. 9, we show how the T2V model's motion priors are strong enough to make a camera rotate around a 3D object, whose position is determined by motion patterns alone.

In Fig. 10 we show how the inference-time hyperparameter γ effects the output videos. Larger γ's allow for a looser control of motion.

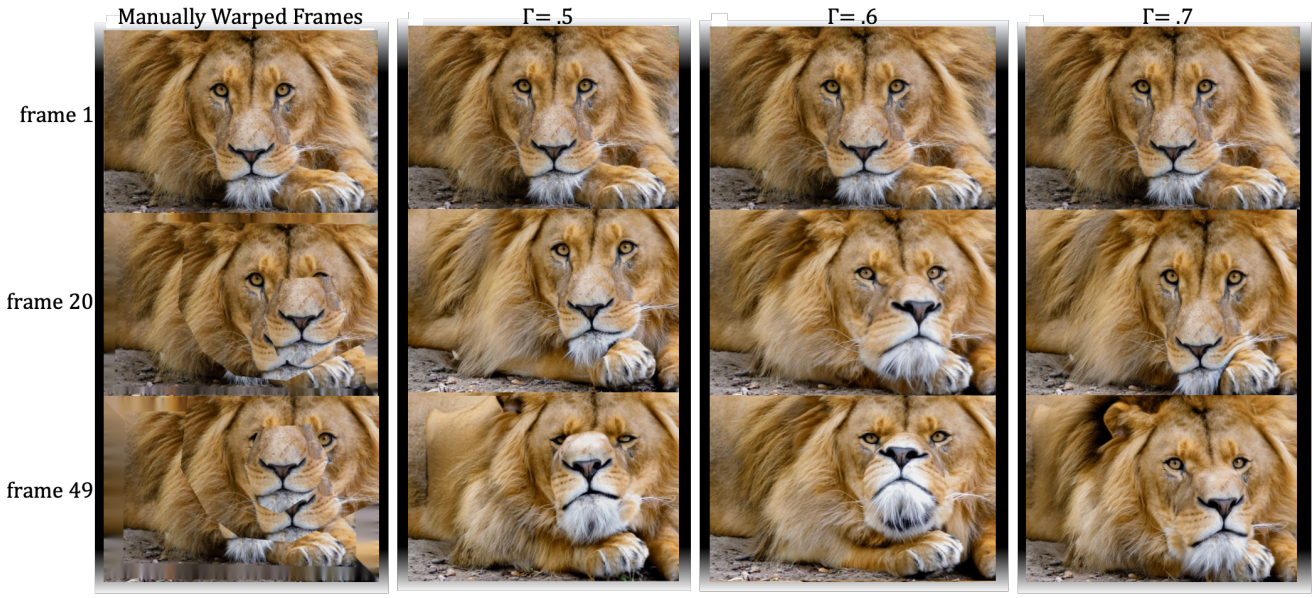


Figure 10. Noise degradation level γ on generated videos. A few frames from the driving video are shown in the leftmost column. Our model outputs are in the next 3 columns. As degradation decreases (γ from 0.7 to 0.5), the video more strictly adheres to the input flow. This allows us to control video movement with a user-specified level of precision.

6.1. Qualitative results of training-free image diffusion based video editing

Noise warping methods that do not preserve Gaussianity degrade per-frame quality, as originally pointed out in [7]. For example, using nearest neighbor and bilinear interpolation destroys the Gaussianity (Fig. 11) and consequently deteriorates the per-frame quality on pre-trained image-to-image diffusion models (Fig. 12 and Fig. 13).

6.2. Comparison to the video diffusion base model without finetuning

Interestingly, video diffusion models respond to noise warping even without training. In Fig. 14 the rightmost column, even though the per-frame quality suffers, the flow of the output video still roughly follows the flow of the warped noise. However, because warped noise is statisically distinct from the pure Gaussian noise CogVideoX was trained on, without fine-tuning it can result in visual artifacts.

6.3. User study settings and statistics

Fig. 15 presents our user study questionnaires and statistics for two applications: (1) local object motion control, and (2) turnable camera movement video generation. Our questions focus on users' overall subjective preference, controllability, and temporal consistency.

6.4. Model Agnostic

Our method is data- and model-agnostic. It can be used to add motion control to arbitrary video diffusion models by only processing the noise sampling during fine-tuning. For example, it also works with AnimateDiff [15] fine-tuned on the WebVid dataset [2] (the weights for this model on our GitHub page). See its qualitative results in Fig. 16. Since release, the community has also trained a version of Go-with-the-Flow on HunyuanVideo (linked on our GitHub page). Therefore, our method will generalize to future more advanced video diffusion base model.

7. The advantage of noise warping

By using noise warping as a condition for motion, we effectively discard all structural information from our input video that cannot be inferred from motion alone. This can be advantageous, as demonstrated in Fig. 14. MotionClone does not use optical flow to guide the video trajectory, instead relying on manipulating activations within the diffusion model. As a result, the windmill gains an extra set of arms, whereas our method, which relies solely on motion information from optical flow via warped noise, does not introduce such artifacts.

8. Conclusion

In this work, we introduce a novel and faster-than-real-time noise warping algorithm that seamlessly incorporates motion control into video diffusion noise sampling, bridging the gap between chaos and order in generative modeling. By leveraging this noise warping technique to preprocess video data for video diffusion fine-tuning, we provide a uni-

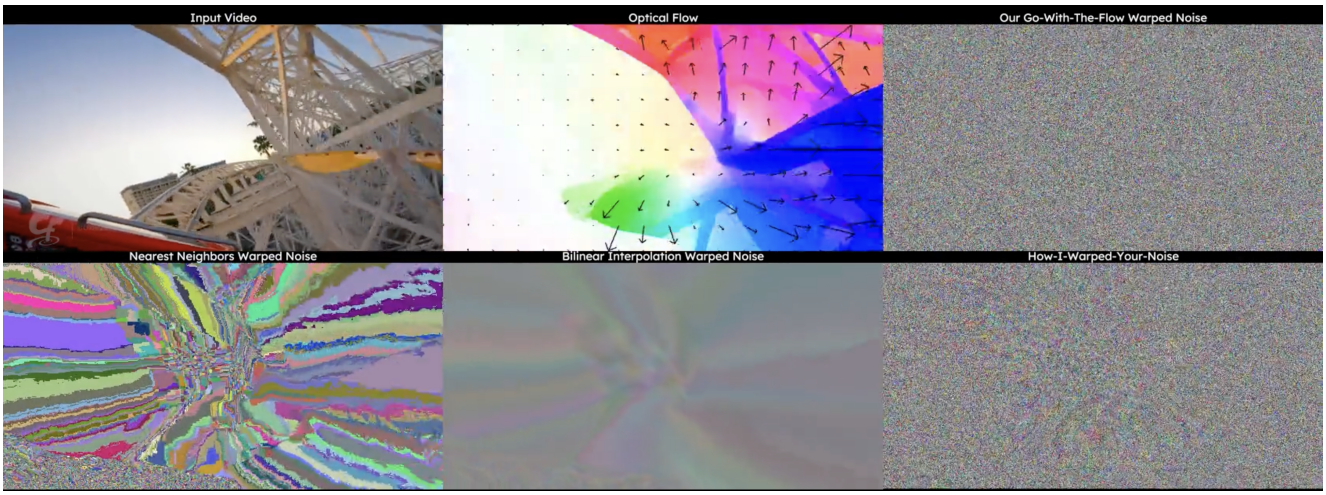


Figure 11. A direct visualization of the noise produced by our noise warping algorithm, HIWYN [7], bilinear, and nearest neighbor interpolations. The forward movement in this long roller-coaster video forces the noise to expand significantly. Early in the video, the HIWYN baseline produces visibly non-Gaussian results. See the full video on our webpage.

Figure 12. Using different noise warping algorithms on DeepFloyd IF for video super-resolution on the DAVIS dataset.

Figure 13. Using different noise warping algorithms on DifFRelight for portrait video relighting.

fied paradigm for a wide range of user-friendly, motion-controllable video generation applications. Extensive experiments and user studies demonstrate the superiority of our method in terms of visual quality, motion controllability, and temporal consistency, making it a robust and versatile solution for motion control in video diffusion models.

9. Social impact statement

Our work contributes to the growing field of video generative models by advancing motion-controllable video generation, which has the potential to revolutionize creative industries such as filmmaking and animation. By introducing a computationally efficient and accessible framework, our method democratizes high-quality video generation, enabling creators, developers, and artists to produce dynamic content with minimal resources or specialized training.

However, we acknowledge the potential misuse of such technology, including the creation of deepfakes or misleading media. To mitigate these risks, we advocate for responsible use, proper content labeling, and the integration of detection mechanisms to ensure ethical deployment. Our approach also emphasizes compatibility with diverse models, encouraging transparency and collaboration within the research community to address societal concerns effectively while maximizing the positive impact of this technology.

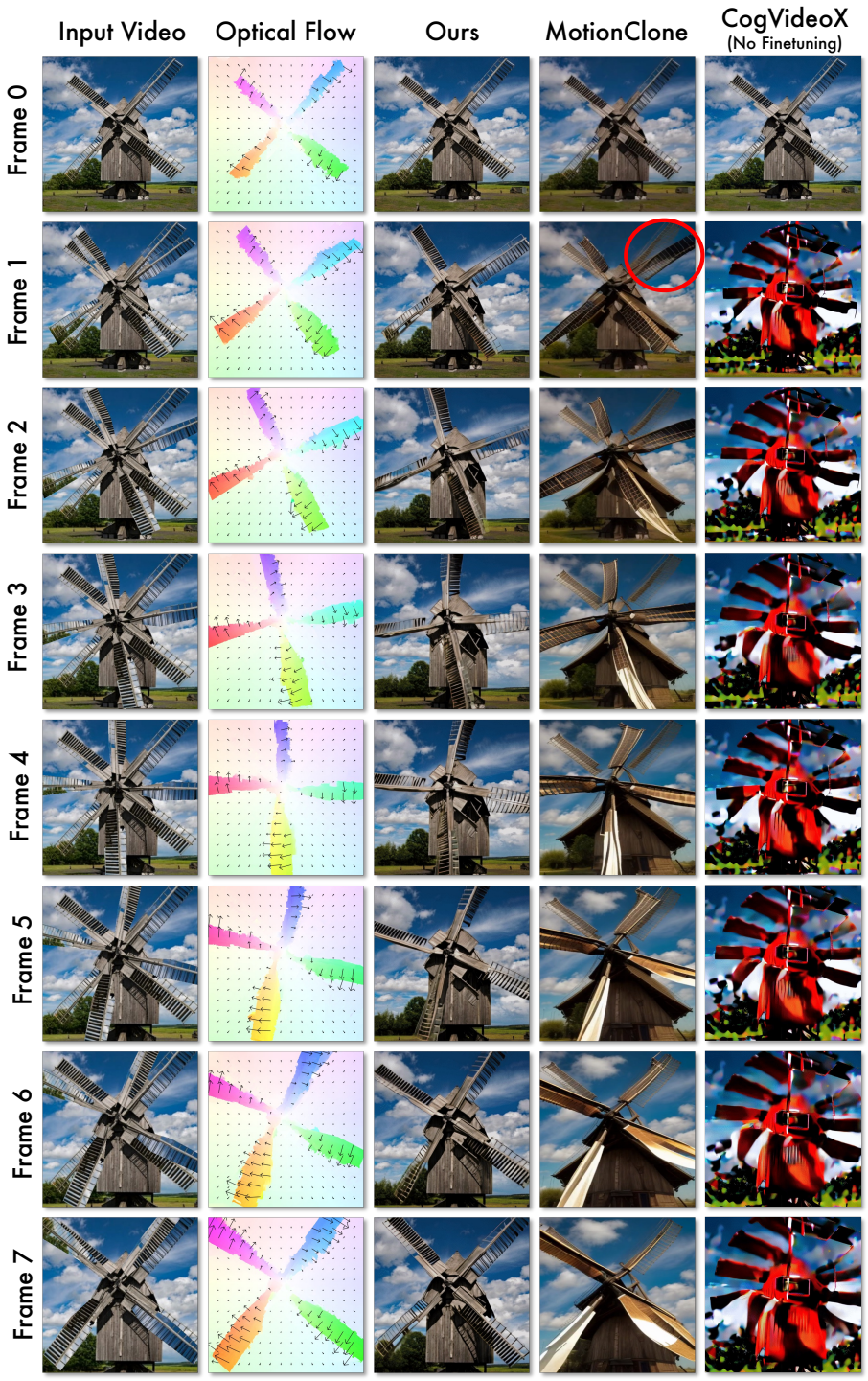
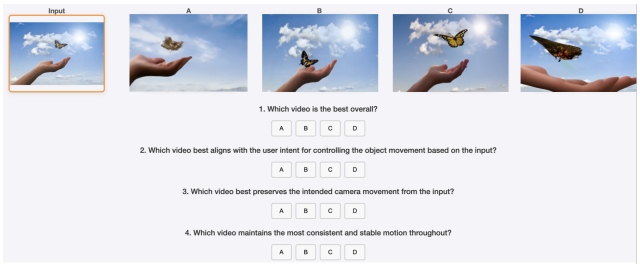
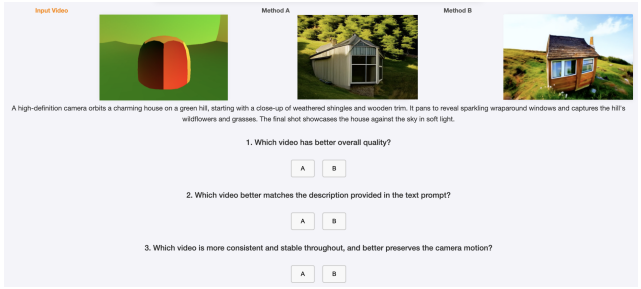


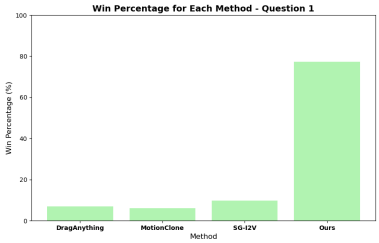
Figure 14. We show a cut-and-drag animation of a windmill rotating clockwise, next to the derived optical flow, our outputs, a baseline and an ablation. **Note** that the input video column appears to have two sets of panels because it's being cut and dragged over itself to create rotational motion. **When using noise warping is better**: Per-frame structural information can poison the result of MotionClone, giving the windmill an extra set of arms - whereas ours only receives motion information from optical flow alone via warped noise (there are no double-windmills in the optical flow patterns). **Ablation in rightmost column**: warped noise with $\gamma = .5$ on the CogVideoX base model before we fine-tune it. Because warped noise is statisically distinct from the pure Gaussian noise CogVideoX was trained on, without fine-tuning it can result in visual artifacts. Note how although the per-frame quality suffers here, it still picks up on motion queues from the warped noise (the camera zooms into the windmill).



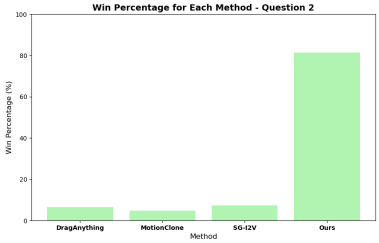
(a) User study interface and questions for local object motion control, corresponding to Fig. 3 in the main paper.



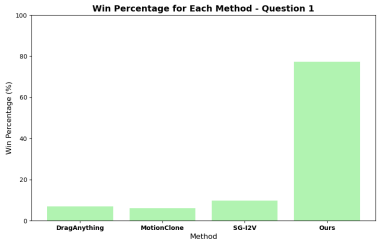
(b) User study interface and questions for turnable camera movement video generation, corresponding to Fig. 9 in the main paper.



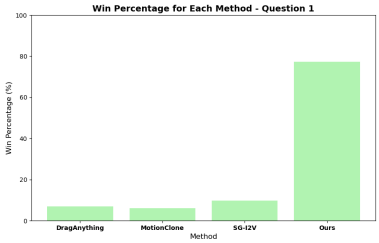
(c) User study statistics for local object motion control on the first question "*Which video is the best overall?*"



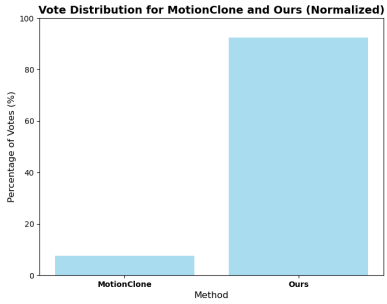
(d) User study statistics for local object motion control on the second question "*Which video best aligns with the user intent for controlling the object movement based on the input?*"



(e) User study statistics for local object motion control on the third question "*Which video best preserves the intended camera movement from the input?*"



(f) User study statistics for local object motion control on the fourth question "*Which video maintains the most consistent and stable motion throughout?*"



(g) User study statistics for motion transfer on the first question "*Which video has better overall quality?*"

Figure 15. User study questionnaires screenshots and statistics. For all the questions of both applications, our method (the rightmost bar plot) significantly wins the most user preferences.

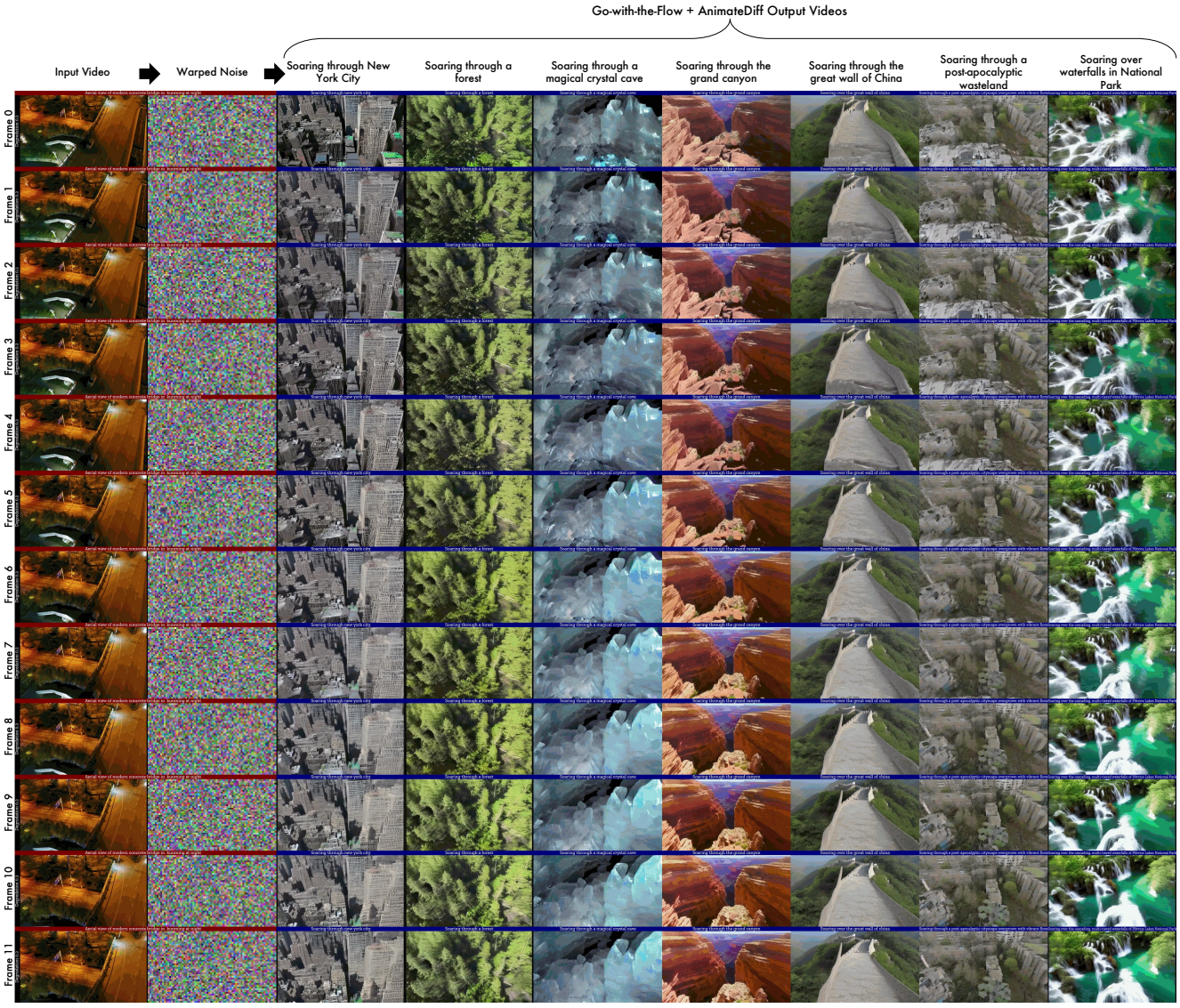


Figure 16. Fine-tuning AnimateDiff with our warped noise flow. We used Go-with-the-Flow to fine-tune AnimateDiff T2V, and display the results above. The input video is on the left, and from that video we derive warped noise which is used to initialize AnimateDiff on the columns to its right with different text prompts.

```
def warp_noise(prev_frame, cur_frame, prev_noise, prev_weight):

    height, width, _ = prev_frame.shape

    flow = optical_flow(prev_frame, cur_frame) # Agnostic to the optical flow algorithm
    backwards_flow = -flow # A cheap approximation of optical_flow(cur_frame, prev_frame)

    expansion_noise    = zeros(height, width)
    contraction_noise  = prev_noise.copy()

    expansion_mask     = ones (height, width, type=bool)
    contraction_mask   = zeros(height, width, type=bool)

    for x in range(width): for y in range(height):
        dx, dy = flow[x,y]
        if 0 <= x+dx <= width-1 and 0 <= y+dy <= height-1:
            # This particle stays in bounds
            expansion_mask  [x+dx, y+dx] = False
            contraction_mask[x   , y   ] = True  # Contraction mask is True where

    for x in range(width): for y in range(height):
        if expansion_mask[x, y]:
            dx, dy = backwards_flow[x,y]
            expansion_noise [x, y] = prev_noise[x+dx, y+dy]

    # We've decided which source pixels are involved in contraction and expansion now
    contraction_noise &= contraction_mask
    expansion_noise, contraction_noise, cur_weight = jointly_regaussianize_and_rebalance_weights(
        expansion_noise, contraction_noise, prev_weight
    ) # Regaussianize all noise values here, and divide the weights by the number of pixels in each bin

    contraction_weight = zeros(height, width)
    for x in range(width): for y in range(height):
        if contraction_mask[x, y]:
            # Contraction treats the noise pixels as particles, each moving from the source to the
            # destination with this flow
            dx, dy = flow[x,y]
            # Contraction is a weighted sum of source pixels to a destination pixel
            pixel_weight = cur_weight[x, y]
            # Sum all the source noise pixels that contract to the same destination
            contraction_noise [x+dx, y+dy] += prev_noise[x, y] * pixel_weight
            # When we multiply a noise pixel by a weight, the variance changes by that weight squared
            contraction_weight[x+dx, y+dy] += pixel_weight ** 2
    contraction_noise /= sqrt(contraction_weight) # Adjust the variance of the summed contracted noise

    # Mixing contraction and expansion noises with their respective masks
    cur_noise = contraction_noise & contraction_mask + expansion_noise & expansion_mask

    return cur_noise, cur_weight
```

Figure 17. Our noise warping pseudo code.