UniVerse: Unleashing the Scene Prior of Video Diffusion Models for Robust Radiance Field Reconstruction

Supplementary Material

7. More Details for Method

7.1. Sorting Images for Sparse Trajectory.

Starting with K poses $\{P_i\}_{i=1}^K$, we initialize a double linked list $\{P_i^l\}_{i=1}^L$ with a randomly chosen pose $P_{\text{init}} \in \{P_i\}_{i=1}^K$, where L is the length of the list. At each iteration, for any pose $P_c \in \{P_i\}_{i=1}^K \setminus \{P_i^l\}_{i=1}^L$, we calculate its distance with the list D_{PL} as follows:

$$D_{PL}(P_c, \{P_i^l\}_{i=1}^L) = \min\{D_P(P_c, P_{head}^l), D_P(P_c, P_{tail}^l)\}. \tag{7}$$

Here, P_{head}^l and P_{tail}^l are the head and tail of the current list, i.e., P_1^l and P_L^l. The distance between poses D_P is defined as:

$$D_P(P_a, P_b) = \frac{\omega_r}{s_R} \cdot D_R(R_a, R_b) + \frac{1-\omega_r}{s_T} \cdot D_T(T_a, T_b). \tag{8}$$

Here, R_a and T_a are the rotation matrix and translation vector of the pose P_a, respectively. ω_r is the weight for rotation distance, and s_R and s_T are scale factors to ensure rotation and translation distances have the same scale. We calculate the rotation distance D_R as:

$$D_R(R_a, R_b) = \arccos\left(\frac{\text{trace}(R_a R_b) - 1}{2}\right), \tag{9}$$

and the translation distance D_T as:

$$D_T(T_a, T_b) = \|T_a - T_b\|_2. \tag{10}$$

After calculating the distances of all P_c and $\{P_i^l\}_{i=1}^L$, we add the new pose P_{new} with minimal distance to the list:

$$P_{new} = \underset{P_c \in \{P_i\}_{i=1}^K \setminus \{P_i^l\}_{i=1}^L}{\arg\min} D_{PL}(P_{new}, \{P_i^l\}_{i=1}^L). \tag{11}$$

If P_{new} is closer to P_{head}^l, we add an edge from P_{head}^l to P_{new} and turn P_{new} into P_{head}^l; otherwise, we do the same for P_{tail}^l. We iteratively perform this process until all poses in $\{P_i\}_{i=1}^K$ are added to the list. After that, we start from P_{head} and traverse the whole list by edges to get an ordered set of poses $\{P_i'\}_{i=1}^K$ (i.e., $\{P_i^l\}_{i=1}^K$).

According to $\{P_i'\}_{i=1}^K$, we obtain the ordered images $\{I_i'\}_{i=1}^K$. Along the ordered poses $P_1', P_2', \ldots, P_K'$, we actually obtain an appropriate implicit camera trajectory. We show this process in both Alg. 2 and Fig. 10.

Algorithm 1 UniVerse

Input: Inconsistent multi-view images $\{I_i\}_{i=1}^K$, rough camera poses $\{P_i\}_{i=1}^K$, camera pose estimation method $Camera(\cdot)$ conditional video diffusion model $\mathcal{V}(\cdot)$, number of images per iteration N, pose sort function $ThreadPose(\cdot)$, the function to turn images to initial videos $I2V(\cdot)$, transient occlusions segment model $Seg(\cdot)$, 3D Reconstruction Method $Recon(\cdot)$

```
1:  **Initialization:** $I^{re\prime} \leftarrow \{\}$
    $\{I_i'\}_{i=1}^K, \{P_i'\}_{i=1}^K \leftarrow ThreadPose(\{I_i\}_{i=1}^K, \{P_i\}_{i=1}^K)$
    $I_{sty} \leftarrow$ manually/random choose an image from $\{I_i'\}_{i=1}^N$
2:  **while** $K > 1$ **do**
3:      Initiate inpainting and style masks $M^{in} \leftarrow \{\}, M^{st} \leftarrow \{\}$
4:      Extract the first $N$ images: $\{I_i'\}_{i=1}^N$
5:      $\mathbf{v}^{ini} \leftarrow I2V(\{I_i'\}_{i=1}^N, f)$, $\mathbf{v}^{ini}$ refers to initial video
6:      **for** each frame $v_j$ in $\mathbf{v}^{ini}$ **do**
7:          **if** $v_j \in \{I_i'\}_{i=1}^N$ **then**
8:              Mask transient occlusions: $M_j^{in}, v_j \leftarrow Seg(v_j)$
9:          **else**
10:             Fill the inpainting mask $M_j^{in}$ with "1"
11:         **end if**
12:         **if** $v_j$ **is** $I_{sty}$, **then**
13:             Fill style mask $M_j^{st}$ with "1".
14:         **else**
15:             Fill style mask $M_j^{st}$ with "0".
16:         **end if**
17:         $M^{in}.append(M_j^{in}), M^{st}.append(M_j^{st})$
18:     **end for**
19:     $\mathbf{v}^{re} \leftarrow \mathcal{V}(\mathbf{v}^{ini}, M^{in}, M^{st})$
20:     Extract the restored images $\{I_i^{re\prime}\}_{i=1}^N$ from $\mathbf{v}^{re}$
21:     $I^{re\prime} \leftarrow I^{re\prime} \cup \{I_i^{re\prime}\}_{i=1}^N$
22:     $I_{sty} \leftarrow I_N^{re\prime}$
23:     $\{I_i'\}_{i=1}^K \leftarrow \{I_i'\}_{i=N+1}^K, K \leftarrow \max(K-N, 0)$
24:     $\{I_i'\}_{i=1}^K \leftarrow \{I_N^{re\prime}\} \cup \{I_i'\}_{i=1}^K$
25:     Update $K \leftarrow K + 1$
26: **end while**
27: # now we get consistent images $I^{re\prime}$ (i.e. $\{I_i^{re\prime}\}_{i=1}^K$)
28: $\{P_i'\}_{i=1}^K \leftarrow Camera(I^{re\prime})$ # estimate poses again using consistent images
29: **Output:** the reconstructed 3D scene $Recon(I^{re\prime}, \{P_i'\}_{i=1}^K)$
```

7.2. Sampling Implicit Views

At each iteration, given N ordered poses $\{P_i'\}_{i=1}^N$ and corresponding N inconsistent images $\{I_i'\}_{i=1}^N$, our goal now is to create a initial video of f frames inluding all the input N images. And we inflate it to f frames by sampling

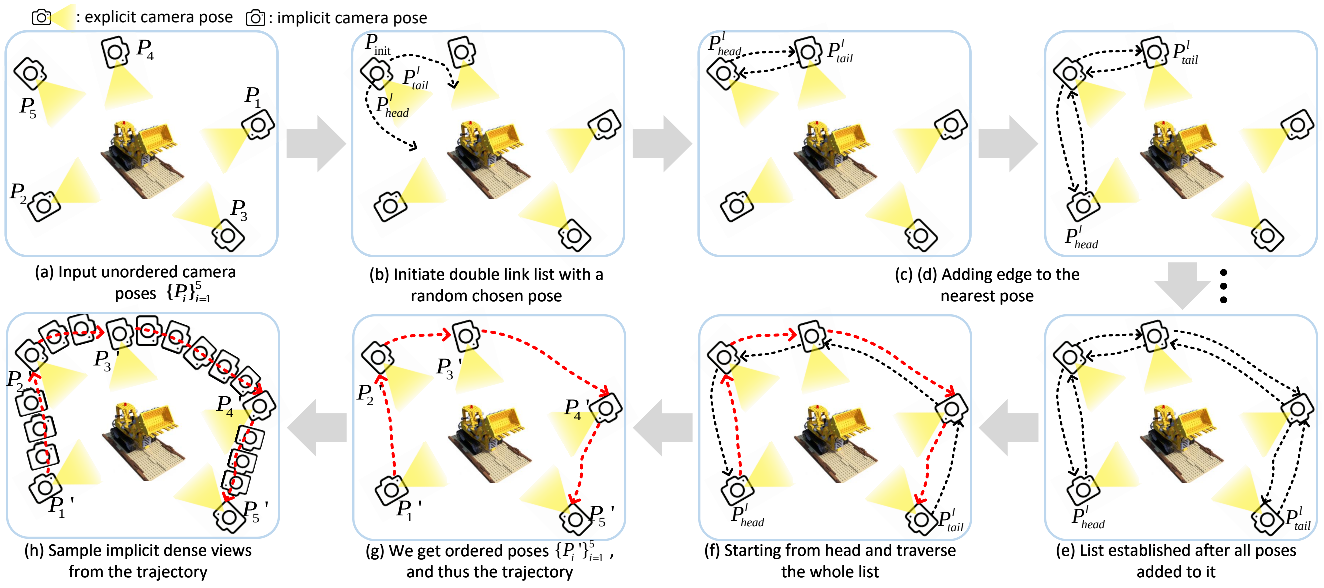
: explicit camera pose
: implicit camera pose
(a) Input unordered camera poses $\{P_i\}_{i=1}^5$
(b) Initiate double link list with a random chosen pose
(c) (d) Adding edge to the nearest pose
(e) List established after all poses added to it
(f) Starting from head and traverse the whole list
(g) We get ordered poses $\{P_i'\}_{i=1}^5$, and thus the trajectory
(h) Sample implicit dense views from the trajectory
P_{init}
P^l_{head}
P^l_{tail}

Figure 10. The flowchart of how we transform a set of multi-view images into a initial video. Here we take an example with 5 input images and their poses. Given 5 unordered poses shown in (a), we firstly random choose a pose to initiate a double link list in (b). Next, we iteratively add the nearest pose to the list until all poses are in the list, shown in (c)(d)(e). Then in (f) we start from the head of the list and traverse the whole list and obtain the ordered poses in (g). Finally we add new poses to the intervals of input poses, making the trajectory dense and thus transform images to video.

Algorithm 2 ThreadPose for Implicit Camera Trajectory

Input: Poses $\{P_i\}_{i=1}^K$, $add\_edge(\cdot)$ func to add bidirectional edges, $Traverse(\cdot)$ func to traverse the list by edges

```
1:  Initialize a double linked list $\{P_i^l\}_{i=1}^L$ with a randomly chosen pose $P_{\text{init}} \in \{P_i\}_{i=1}^K$
2:  Set $L \leftarrow 1$, $P_1^l \leftarrow P_{\text{init}}$
3:  **while** $\{P_i\}_{i=1}^K \setminus \{P_i^l\}_{i=1}^L$ is **not empty do**
4:      Find the pose $P_{new}$ with the minimal distance:
            $$P_{new} = \underset{P_c \in \{P_i\}_{i=1}^K \setminus \{P_i^l\}_{i=1}^L}{\arg\min} D_{PL}(P_{new}, \{P_i^l\}_{i=1}^L)$$
5:      **if** $D_P(P_{new}, P^l_{head}) < D_P(P_{new}, P^l_{tail})$ **then**
6:          $P_{head}.add\_edge(P_{new})$
7:          $P^l_{head} \leftarrow P_{new}$
8:      **else**
9:          $P_{tail}.add\_edge(P_{new})$
10:         $P^l_{tail} \leftarrow P_{new}$
11:     **end if**
12:     $L \leftarrow L + 1$
13: **end while**
14: $\{P_i'\}_{i=1}^K \leftarrow Traverse(P_{head})$
15: **Output:** Ordered poses $\{P_i'\}_{i=1}^K$
```

$f - N$ new poses and thus new views. First, we compute the distances $\{d_i\}_{i=1}^{N-1}$ between neighboring poses:

$$d_i = D_P(P_i', P_{i+1}'), \quad i = 1, 2, \dots, N-1. \tag{12}$$

Next, we determine the number of new poses n_i to be inserted between each pair of neighboring poses P_i' and P_{i+1}', proportional to the distance d_i:

$$n_i = \left\lfloor \frac{d_i}{\sum_{i=1}^{N-1} d_i} \times (f - N) \right\rfloor, \quad i = 1, 2, \dots, N-1, \tag{13}$$

where $\lfloor x \rfloor$ denotes the floor function, which gives the greatest integer $\leq x$. Since the sum of n_i might not exactly equal $f - N$ due to the floor operation, we distribute the remaining poses. We calculate the remaining number of poses r:

$$r = (f - N) - \sum_{i=1}^{N-1} n_i. \tag{14}$$

Then, we add one additional pose to the r largest intervals (i.e., the intervals with the largest d_i values) by incrementing n_i for the r largest d_i values:

$$n_i = n_i + \begin{cases} 1 & \text{if } d_i \text{ is among the } r \text{ largest values,} \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

In this way, we obtain the number of inserted views. By inserting n_i zero frames into neighboring images I_i', I_{i+1}', we get the initial video.

8. More Implementation Details

Adapt Video Diffusion Models with Mask Input: We fine-tune the Video Diffusion Model from the 576×1024 interpolation model of ViewCrafter [67]. Since our method utilizes additional masks (*i.e.* inpainting masks and style masks), we need to change the input dimension of the Denoising U-Net. We follow the fine-tuning approach of Inpainting Latent Diffusion [37]. Specifically, we change an $8 \times C \times \textit{kernel\_size} \times \textit{kernel\_size}$ 2D convolutional kernel to $10 \times C \times \textit{kernel\_size} \times \textit{kernel\_size}$ by concatenating two additional masks. To do this, we maintain the original $8 \times C \times \textit{kernel\_size} \times \textit{kernel\_size}$ kernels and add zero-initialized $2 \times C \times \textit{kernel\_size} \times \textit{kernel\_size}$ kernels to it.

Detect All Transient Objects in Input Images: In the UniVerse pipeline, it is important to identify all transient objects to mask them. To achieve this, we first pre-define a set of transient prompts, such as `[person, car, bike]`. We then use a Semantic Segmentation Model to detect the pixels of the objects in the prompts. Using the positions of these pixels, we employ the Segment Anything Model (SAM) [24] to precisely segment the objects and obtain the inpainting masks.

9. More Visual Results

Since UniVerse utilizes a VDM to turn initial videos into restored videos, we present several examples in Figs. 11, 12, and 13, demonstrating how UniVerse leverages the video prior to transform multi-view images into a consistent video. In these figures, the top row shows the initial video frames, while the bottom row displays the corresponding restored video frames. The frames are arranged from left to right in sequential order, with the first row showing frames 1-5, the second row showing frames 6-10, and so on.

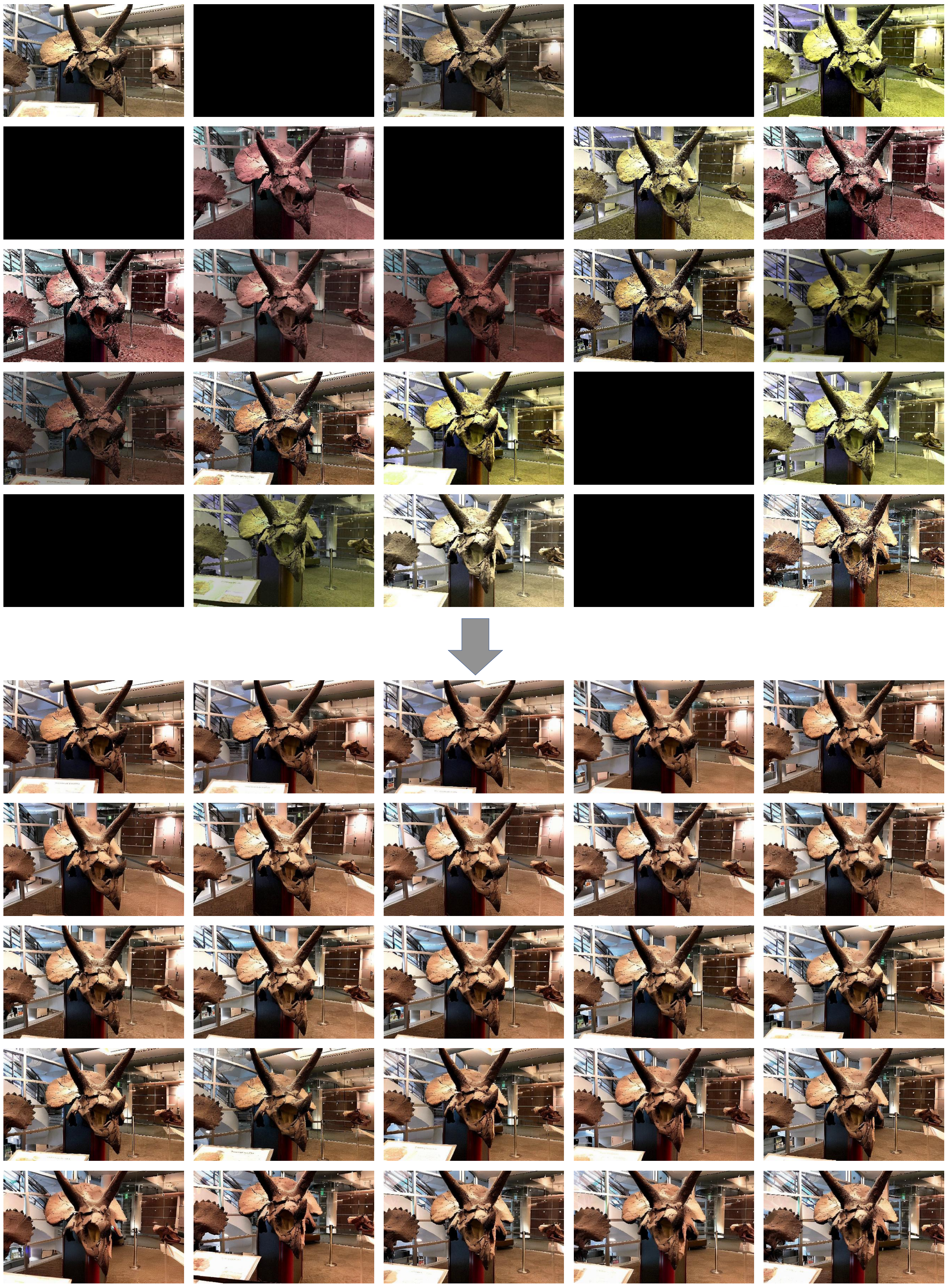

Figure 11. Visualization of how UniVerse turns a initial video into restored video.

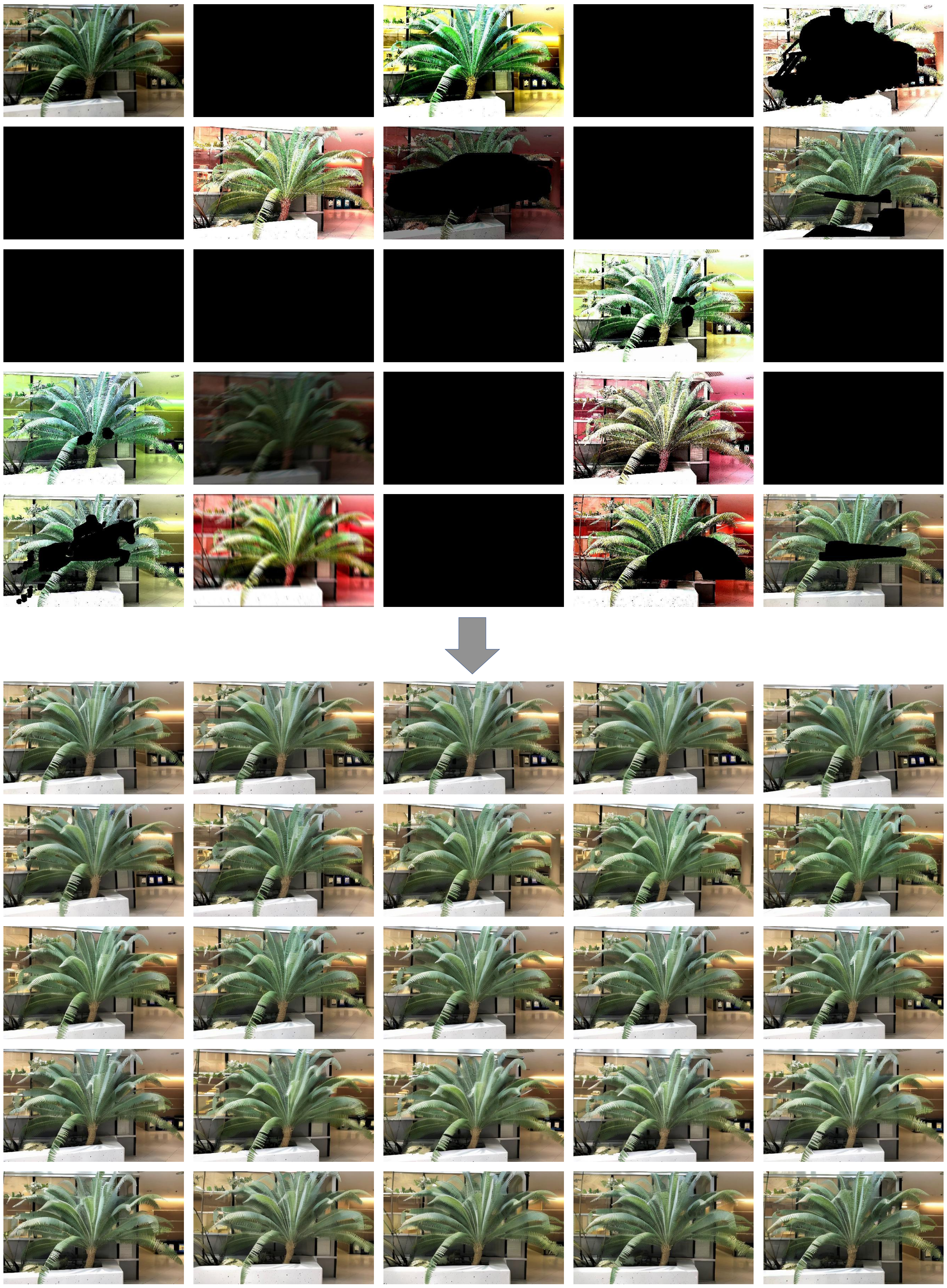
Figure 12. Visualization of how UniVerse turns a initial video into restored video.

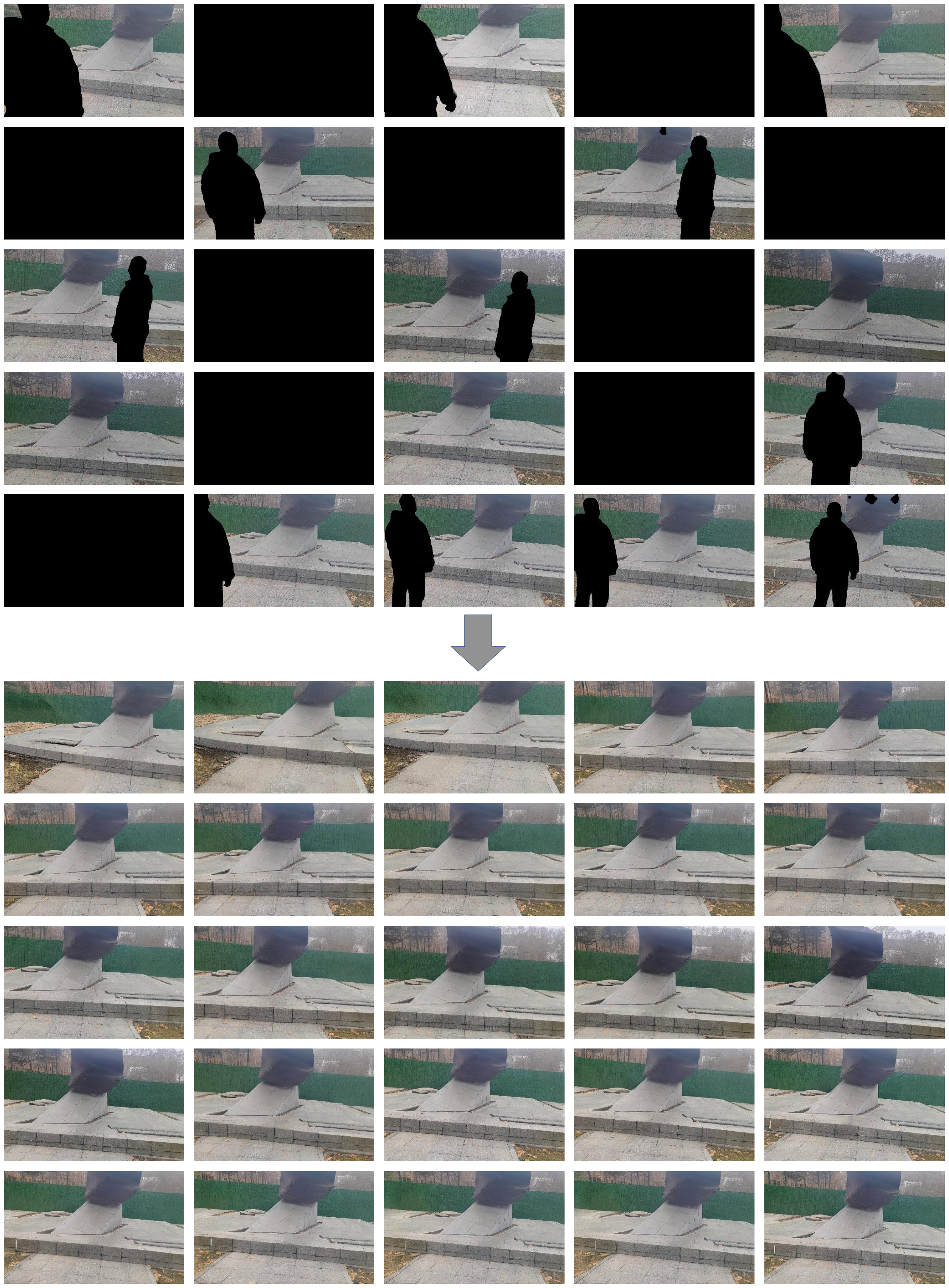
Figure 13. Visualization of how UniVerse turns a initial video into restored video.