Sprite-from-Sprite: Cartoon Animation Decomposition with Self-supervised Sprite Estimation

LVMIN ZHANG, The Chinese University of Hong Kong / Style2Paints, China
TIEN-TSIN WONG, The Chinese University of Hong Kong, Hong Kong, China
YUXIN LIU, The Chinese University of Hong Kong, Hong Kong, China

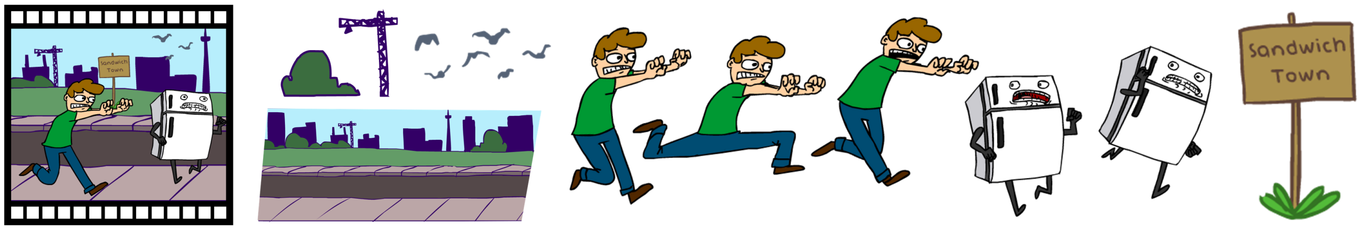


Fig. 1. Given the input cartoon animation on the left, our approach decompose it into several sprites. These sprites can be composed to reconstruct the input animation. The sprites may have multiple frames; see also the video for all frames of these sprites. ©*Sandwich Town, used with artist permission.*

We present an approach to decompose cartoon animation videos into a set of "sprites" — the basic units of digital cartoons that depict the contents and transforms of each animated object. The sprites in real-world cartoons are unique: artists may draw arbitrary sprite animations for expressiveness, where the animated content is often complicated, irregular, and challenging; alternatively, artists may also reduce their workload by tweening and adjusting sprites, or even reuse static sprites, in which case the transformations are relatively regular and simple. Based on these observations, we propose a sprite decomposition framework using Pixel Multilayer Perceptrons (Pixel MLPs) where the estimation of each sprite is conditioned on and guided by all other sprites. In this way, once those relatively regular and simple sprites are resolved, the decomposition of the remaining "challenging" sprites can simplified and eased with the guidance of other sprites. We call this method "sprite-from-sprite" cartoon decomposition. We study ablative architectures of our framework, and the user study demonstrates that our results are the most preferred ones in 19/20 cases.

1 INTRODUCTION

In the era of digital content creation, the production of cartoon animation has moved from paper work to computer software. Computer-aided workflows allow artists to manage their created contents in an advanced way, and the basic unit of animation management is typically called "sprite". Each sprite contains a transparent animation clip, and supports transformations such as panning, rotating, scaling, *etc.* The assistance of animated sprites enhances the reusability of drawing contents and reduces the workload of artists. Many cartoon animation techniques are based on sprites, *e.g.*, sprite tweening, sprite filtering, *etc.* How to decompose a video production of cartoon animation into editable and reusable source sprites is a long-standing and highly demanded problem.

In real-world cartoon animation, sprites are unique in appearance. In some cases, artists can draw arbitrary sprite animations to enhance the visual expression. These animations are often highly complicated, do not follow real physical principles, and sometimes even omit or alter parts of the subject to emphasize the dynamics of characters. For example, in the boy (fig. 2) sprite, the boy's body, hair, and hands have no fixed shape, making precise pixel correlations difficult to establish. The artist's exaggerated animation, which is almost irregular, brings out the boy's lively demeanor. In other cases, on the contrary, the artist may reuse sprites and use software to automatically compute intermediate animations or even directly copy static sprites to save workload and cost. These computed animations are usually simple and regular, and the motion of these sprites can be analyzed and tracked relatively well. For example, the fish and bubble (fig. 2) animations are obtained by moving the respective sprites, while the sea (fig. 2) sprite is a static background. For these relatively simple sprite animations, computational models can routinely fit accurate and reliable motion patterns.

Observing the uniqueness of these sprites, we naturally come up with a question: *can we use some relatively simple sprites to help decompose those more "challenging" sprites?* Since the composition of all sprites as a whole must reconstruct a complete cartoon animation frame, if some of the sprites are resolved, then the decomposition problem of the remaining sprites will be simplified. As shown in figure 3, despite the complexity of the boy animation, if we can guide the boy decomposition with other simple and regular sprites like the fish, bubble, and sea, the exact content of our wanted sprite can be obtained by solving a permutation/combination problem.

To this end, we propose an animation decomposition framework that can automatically discover some sprites with relatively simple and regular animation patterns, and at the same time, can decompose other complicated sprites by learning a "sprite-from-sprite" mapping, where the decomposition of those relatively challenging sprites can be guided by the other sprites. We would like to point out that our framework treats all sprites equally and optimizes them together as a whole, avoiding hard threshold or explicit categorization of sprites as challenging or simple. The core mapping is learned using a Pixel Multilayer Perceptron (Pixel MLP, or called Implicit MLP), which is a reliable component and has shown success in fields like 3D representation (NeRF, [Mildenhall et al. 2020]).

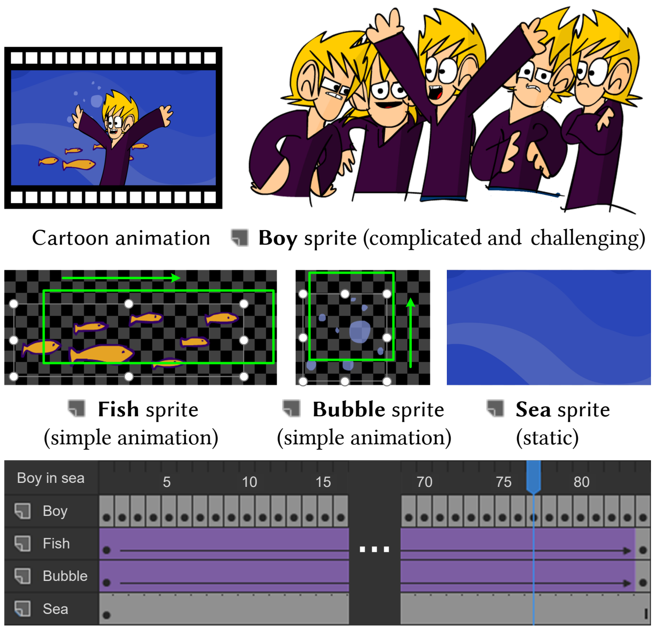

Cartoon animation
Boy sprite (complicated and challenging)
Fish sprite (simple animation)
Bubble sprite (simple animation)
Sea sprite (static)
Boy in sea
5
10
15
70
75
80
Boy
Fish
Bubble
Sea

Fig. 2. **Real-world example of cartoon animation:** We show how cartoon sprites are composed for animation, and the difference in complexity between these sprites. In the timeline below, the black dot indicates an editing keyframe; the purple blocks and arrows indicate the intermediate animations that are automatically computed by the software. ©*Boy in The Sea, used with artist permission.*

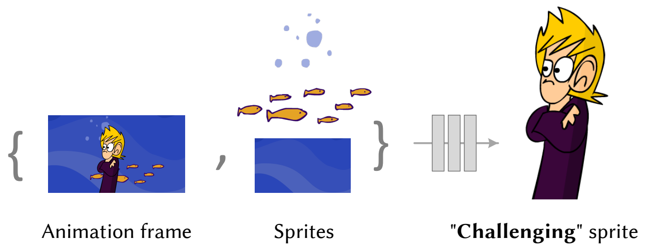


Fig. 3. **Sprite-from-sprite estimation:** We show that the extraction of one sprite can be guided by other sprites so that the decomposition of some "challenging" sprites can be eased. ©*Boy in The Sea, used with artist permission.*

Experimental results demonstrate that a "sprite-from-sprite" decomposition leads to sharp and clean results, as the coverage area of each sprite is optimized with the guidance of other sprites. Furthermore, compared to mainstream cartoon animation processing methods based on detecting cartoon edge lines or color blocks, our framework shows robustness to in-the-wild cartoon animations since the optimization is performed on a per-pixel learning basis. We also present various practical applications using our decomposed sprites. Finally, the user study show that our results are the most preferred ones in 19/20 cases.

In summary, (1) we observe the real sprites of in-the-wild cartoon animations and point out their uniqueness: artists can draw sprites with arbitrary and complicated animation for visual expression, or use software to automatically compute and tween relatively simple animations; (2) Based on these observations we propose a "sprite-from-sprite" framework to decompose cartoon animations into sprites, where the estimation of each sprite is guided by all other sprites, *e.g.*, the extraction of some complicated sprites can be guided by other relatively simple sprites; (3) we demonstrate a variety of cartoon animation applications using our decomposed sprites; (4) the user study validate that, in 19/20 tests, our framework is the most preferred one.

2 RELATED WORK

The decomposition of cartoon animation is different from typical video decomposition tasks. The appearance of cartoon animation poses unique challenges to the decomposition methods, such as abrupt transitions and exaggerations for artistic effect. These special variations will often require methods that rely relatively less on assumptions like the presence of pixel correspondences, invariant illuminations, *etc.*

Cartoon animation decomposition. The decomposition of cartoon animation is a long-standing problem. Traditional methods typically use cartoon edge line patterns and region extraction to establish temporal correspondences. *Globally Optimal Toon Tracking* [Zhu et al. 2016] detects cartoon regions and then solve the temporal tracking; A similar pre-processing region extraction step is proposed in [Liu et al. 2013]. *Vectorizing Cartoon Animations* [Zhang et al. 2009] use flood filling to achieve cartoon color blocks. The usage of cartoon features is also extensively discussed in *Sketching Cartoons by Example* [Sýkora et al. 2005] and *TexToons* [Sýkora et al. 2011]. Recent cartoon techniques also opt flow or dense correspondence like *ToonSynth* [Dvorožňák et al. 2018] and *EbSynth* [Jamriška et al. 2019]. *Autocomplete Hand-drawn Animations* [Xing et al. 2015] use line-drawing-based correspondence to analyze the key-frame motions. The performance of methods using cartoon edge lines or regions depend on the accuracy of the line or region extraction.

Video decomposition. Machine learning has facilitated recent advances in video decomposition. *Omnimatte* [Lu et al. 2021] decomposes input video sequences by training neural networks to reconstruct the scene. *Layered Neural Rendering* [Lu et al. 2020] is oriented to decompose people in videos and trains neural networks to reconstruct backgrounds and human body parts. Another type of approaches views input videos as compositions of multiple warped 2D textures. A typical example is *Unwrap Mosaics* [Rav-Acha et al. 2008] that warps multiple 2D image textures to reconstruct a video for manipulation purposes. *Layered Neural Atlases* [Kasten et al. 2021] trains neural networks to learn such warping and *Deformable Sprites* [Ye et al. 2022] extends this architecture. More generally, [Lao and Sundaramoorthi 2018] addresses 3D motions using layered model; [Ost et al. 2021; Yu et al. 2022] explores 3D object discovery in various settings. We will discuss approaches of this category in our comparisons.

Besides implicit neural representation, self-supervised neural decomposition of videos is studied in various ways. MarioNette [Smirnov et al. 2021] is a deep learning approach that decomposes sprite-based video animations into a disentangled representation of recurring

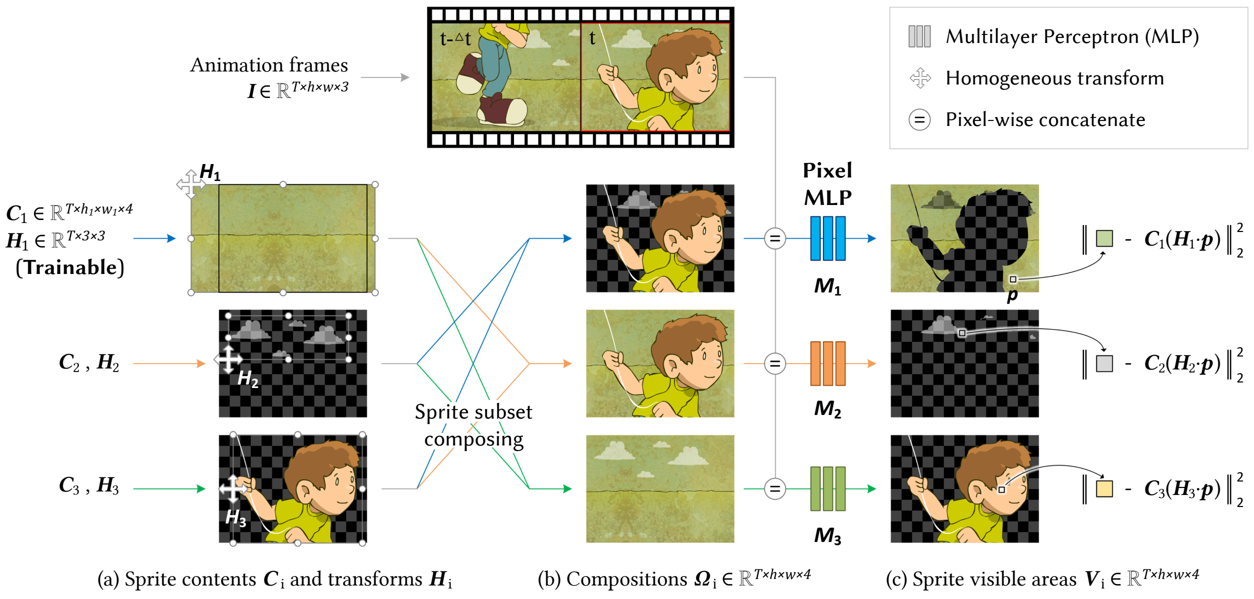

Animation frames
$I \in \mathbb{R}^{T\times h\times w\times 3}$
t-Δt
t
Multilayer Perceptron (MLP)
Homogeneous transform
Pixel-wise concatenate
$C_1 \in \mathbb{R}^{T\times h_1\times w_1\times 4}$
$H_1 \in \mathbb{R}^{T\times 3\times 3}$
(Trainable)
H_1
C_2, H_2
H_2
C_3, H_3
H_3
Sprite subset composing
Pixel MLP
M_1
M_2
M_3
p
$\| \cdot - C_1(H_1 \cdot p) \|_2^2$
$\| \cdot - C_2(H_2 \cdot p) \|_2^2$
$\| \cdot - C_3(H_3 \cdot p) \|_2^2$
(a) Sprite contents C_i and transforms H_i
(b) Compositions $\Omega_i \in \mathbb{R}^{T\times h\times w\times 4}$
(c) Sprite visible areas $V_i \in \mathbb{R}^{T\times h\times w\times 4}$

Fig. 4. **Overview:** In this example, our framework decomposes a cartoon animation into 3 sprites by optimizing 3 sprite content matrix $C_{1...3}$, 3 sprite transform matrix $H_{1...3}$ and 3 pixel MLPs $M_{1...3}$. The number of sprites can be different in other cases. ©*Boy flying a kite, used with artist permission.*

graphic elements in a self-supervised manner. DTI-Sprites [Monnier et al. 2021] represent video objects as explicit transformations of a small set of prototypical images, and then train neural networks to learn the representations. We will include these methods in experiments.

Motion decomposition. Videos can also be decomposed using motions. [Shi and Malik 1998] proposes a spatio-temporal image clustering algorithm. The clustering of optical flow or motion cues are also extensively discussed in [Brox and Malik 2010; Keuper 2017; Keuper et al. 2015; Ochs and Brox 2011; Ochs et al. 2014a]. Recent video object segmentation approaches also show that the pixel-wise flow and motions can be used in learning the segmentation of moving objects [Dave et al. 2019; Xie et al. 2019; Yang et al. 2019]. As we discussed before, the motions in cartoon animations can be arbitrarily drawn by artists. The detected cues of motions is not always accurate and needs to be partially trusted.

Layer decomposition. The functionality of decomposed layers depends on the problem formulation of layer cues, *e.g.*, local similarity, color geometry, illumination, *etc.* Lin *et al.* [2017] propose to fully decompose images into several component layers. The adaptive image decomposition based on color segmentation is discussed in [Aksoy et al. 2017]. Zhang *et al.* [2017] propose to optimize the decomposition to re-color images. Chang *et al.* [2015] propose an editable palette-based layer extraction. Chang *et al.* [2015] re-colorize photo using palette clustering. The grouped image color theme can also be manipulated in [Nguyen et al. 2017]. Tan *et al.* [2017; 2018] show that all pixels in an image can be reproduced by *mixing* the color vertices on the RGB convex hull, where they decompose image into layers using the vertex colors.

Neural representation and Multilayer Perceptron. Pixel MLPs (or called Coordinate-based MLPs, Implicit MLPs, Implicit Neural Representations) show advanced performance in representing 3D geometry [Groueix et al. 2018; Mescheder et al. 2019; Mildenhall et al. 2020; Park et al. 2019] and dynamic video scenes [Li et al. 2020]. Such representations also shows that the complicated mappings in 2D spaces can also be fitted in pixel-wise, even without learning local patterns, *e.g.*, [Tancik et al. 2020], SIRENs [Sitzmann et al. 2020], *etc.* Our approach is based on pixel-wise learning to facilitate accurate decomposition.

3 APPROACH

We present an overview of our framework in figure 4. Our approach decomposes an input cartoon animation into several "sprites" with independent contents and transforms. The decomposition of each sprite is guided by all other sprites, so that the sprite extraction against complicated and irregular motions or appearances can be eased once some relatively simple sprites are solved. We first introduce the framework architecture in Section 3.1, and then describe the objective of our framework in Section 3.2. The training details are presented in Section 3.3.

3.1 Sprite-from-Sprite Decomposition

Given an input cartoon animation RGB video $I \in \mathbb{R}^{T\times w\times h\times 3}$ with T frames and $w \times h$ size, our approach outputs N sprites (fig. 4a), each with a content matrix $C_i \in \mathbb{R}^{T\times w_i\times h_i\times 4}$ and a transform matrix $H_i \in \mathbb{R}^{T\times 3\times 3}$, where $i = 1...N$. The content matrix C_i is a RGBA transparent video with independent width w_i and height h_i; the transform matrix H_i is a sequence of 3×3 homogeneous transforms. Both the content C_i and the transform H_i are learnable parameters.

The homogeneous transform is consistent to the standard of most commercial animation software (*e.g.*, Adobe Animate) for representing various projections like rotating, scaling, translating, shearing, *etc.* Given the transformation $\boldsymbol{H}_{i,t} \in \mathbb{R}^{3\times3}$ (t is frame index) and a pixel position $\boldsymbol{p} = [\boldsymbol{p}_t, \boldsymbol{p}_x, \boldsymbol{p}_y]^\intercal \in \mathbb{R}^3$, we compute the transformed $\boldsymbol{p}' = [\boldsymbol{p}_t, \boldsymbol{p}'_x, \boldsymbol{p}'_y]^\intercal$ with

$$\boldsymbol{p}'_x = \frac{u}{f},\ \boldsymbol{p}'_y = \frac{v}{f} \quad \text{where} \quad [u,\ v,\ f]^\intercal = \boldsymbol{H}_{i,t}[\boldsymbol{p}_x, \boldsymbol{p}_y,\ 1]^\intercal \tag{1}$$

and note that this transform do not influence the temporal index $\boldsymbol{p}_t$. We define that the matrix $\boldsymbol{H}_{i,t}$ projects any pixel position $\boldsymbol{p}$ from the screen space (coordinates of $\boldsymbol{I}$) to the sprite space (coordinates of $\boldsymbol{C}_i$), simplified as $\boldsymbol{p}' = \boldsymbol{H}_{i,t} \cdot \boldsymbol{p}$. We use a pixel interpolation function ϕ (details in Sec. 3.3) to obtain the transformed appearance of sprites as $\phi(\boldsymbol{C}_i, \boldsymbol{H}_{i,t} \cdot \boldsymbol{p})$, abbreviated as $\boldsymbol{C}_i(\boldsymbol{H}_i \cdot \boldsymbol{p})$.

During training, we compose the sprites to obtain N "subset" compositions (fig. 4b). These compositions will be fed into neural networks so that the learning of each sprite can be guided by all other sprites. Denoting that $(\cdot)_\alpha$ gets the alpha opacity of a RGBA transparent color, we use back-to-front alpha blending with

$$\Omega_{i,\boldsymbol{p}} = \sum_{j\neq i} \boldsymbol{C}_j(\boldsymbol{H}_j \cdot \boldsymbol{p})(\boldsymbol{C}_j(\boldsymbol{H}_j \cdot \boldsymbol{p}))_\alpha \prod_{k\neq i,k<j} (1 - (\boldsymbol{C}_k(\boldsymbol{H}_k \cdot \boldsymbol{p}))_\alpha) \tag{2}$$

where $\Omega_i \in \mathbb{R}^{T\times w\times h\times 4}$ (fig. 4b) is one computed composition. For each sprite, we train a pixel Multilayer Perceptron (pixel MLP) to guide the sprite decomposition using the mapping

$$\boldsymbol{M}_i : \ [\boldsymbol{p}\ \ \boldsymbol{I}_{\boldsymbol{p}}\ \ \Omega_{i,\boldsymbol{p}}]^\intercal \to \boldsymbol{V}_{i,\boldsymbol{p}} \tag{3}$$

where $\boldsymbol{M}_i : \mathbb{R}^{10} \to \mathbb{R}^4$ is a trainable mapping that extracts the sprite visible areas $\boldsymbol{V}_i \in \mathbb{R}^{T\times w\times h\times 4}$ in each frame (fig. 4c). Each pixel value of $\boldsymbol{V}_i$ is processed independently. Each pixel MLP $\boldsymbol{M}_i$ has independent weights.

In this way, the learning of each sprite is conditioned on and guided by all other sprites. Although all sprites are optimized together, the interaction between the sprites (the contrast between sprites) is important for achieving high-quality results. Because both the input and output of the MLP are sprites, we call this method sprite-from-sprite decomposition. Besides, since we treat all sprites equally, we do not need to explicit classify the sprites into "easy" or "challenging" ones. This avoids artifacts caused by hard thresholds and classifications.

We jointly train this fully-differentiable framework so that the sprite visible areas $\boldsymbol{V}_i$ can reconstruct the original video and the sprites contents and transforms $\{\boldsymbol{C}_i, \boldsymbol{H}_i\}$ can memorize and reproduce those visible areas in a consistent way. The learning objective can be written as

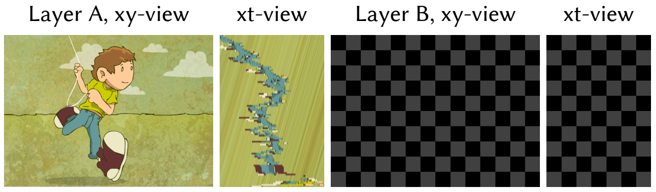
$$\mathcal{L} = \alpha_{\text{rec}}\mathcal{L}_{\text{rec}} + \alpha_{\text{mem}}\mathcal{L}_{\text{mem}} + \alpha_{\text{occ}}\mathcal{L}_{\text{occ}} + \alpha_{\text{tv}}\mathcal{L}_{\text{tv}} + \alpha_{\text{flow}}\mathcal{L}_{\text{flow}} \tag{4}$$

where $\alpha_{\ldots}$ values are balancing parameters. We next detail these objectives.

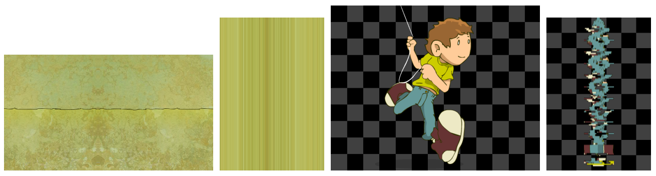
3.2 Learning Objective

Reconstruction consistency. The reconstruction of the original cartoon animation is given as

$$\mathcal{L}_{\text{rec}} = \alpha_{\text{color}}\mathcal{L}_{\text{color}} + \alpha_{\text{alpha}}\mathcal{L}_{\text{alpha}} \tag{5}$$



(a) Without temporal variation minimization



(b) With temporal TV-L1 minimization

Fig. 5. **Minimizing temporal variation:** We present two sprites extracted by our framework with or without temporal regulation. ©*Boy flying a kite, used with artist permission.*

where α_{color} and α_{alpha} are weights. The input cartoon video has to be reconstructed by the visible parts of all sprites as

$$\mathcal{L}_{\text{color}} = ||\boldsymbol{I}_{\boldsymbol{p}} - \sum_i (\boldsymbol{V}_{i,\boldsymbol{p}})_{\text{RGB}}(\boldsymbol{V}_{i,\boldsymbol{p}})_\alpha||_2^2 \tag{6}$$

where $(\cdot)_{\text{RGB}}$ and $(\cdot)_\alpha$ gets the RGB color and alpha weight. The alpha weights need to sum up as one

$$\mathcal{L}_{\text{alpha}} = ||1 - \sum_i (\boldsymbol{V}_{i,\boldsymbol{p}})_\alpha||_2^2 \tag{7}$$

where we note that the blending is order-invariant (different from back-to-front blending). We view the $(\boldsymbol{V}_{i,\boldsymbol{p}})_\alpha$ as a soft estimation to what extent the i-th sprite is visible in the screen.

Memory consistency. We encourage the sprite contents and transforms to memorize the sprite visible areas in video frames

$$\mathcal{L}_{\text{mem}} = \overline{\boldsymbol{O}}_{i,\boldsymbol{p}}||\boldsymbol{V}_{i,\boldsymbol{p}} - \boldsymbol{C}_i(\boldsymbol{H}_i \cdot \boldsymbol{p})||_2^2 \tag{8}$$

where $\overline{\boldsymbol{O}}_i \in \mathbb{R}^{T\times w\times h}$ is an inverted occlusion map (occluded pixels are zeros and other pixels are ones). This map is computed as a soft alpha weight summing up of all visible sprites above the i-th sprite

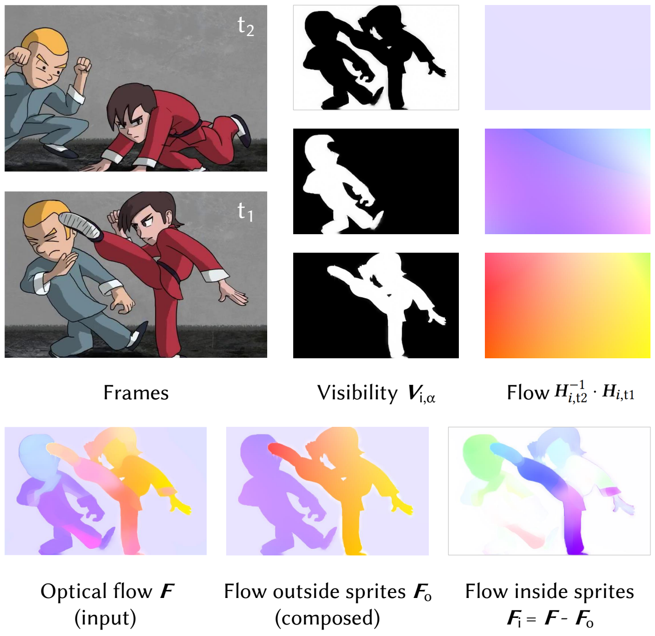
$$\overline{\boldsymbol{O}}_{i,\boldsymbol{p}} = 1 - \sum_{k>i} (\boldsymbol{V}_{k,\boldsymbol{p}})_\alpha \tag{9}$$

and we note that all the non-occluded colors and alpha weights in $\boldsymbol{V}_i$ are memorized by the sprites as contents and opacity.

Temporal consistency. If without any temporal regulation, the framework may cheat with some trivial solutions, *e.g.*, one sprite copying the input video and all other sprites being blank (fig. 5a). We regulate the sprites with a temporal TV-L1 denoising

$$\mathcal{L}_{\text{tv}} = ||\nabla_{\text{t}}\boldsymbol{C}_{i,\boldsymbol{p}}||_1 \tag{10}$$

where ∇_{t} computes the matrix gradient over the t-axis. This regulation encourages temporal consistency inside each sprites (fig. 5b). In extreme cases, *e.g.*, when the sprite animation only contains transformations of a static image, the temporal variation $\mathcal{L}_{\text{tv}}$ can



t2
t1
Frames
Visibility $V_{i,\alpha}$
Flow $H_{i,t2}^{-1} \cdot H_{i,t1}$
Optical flow F (input)
Flow outside sprites F_o (composed)
Flow inside sprites $F_i = F - F_o$

Fig. 6. **Decomposing optical flow:** We decompose an input optical flow into a flow outside sprites and a flow inside sprites. ©*Kung Fu, used with artist permission.*

be minimized to zero. In other cases, the minimization facilitates object alignment inside sprites.

Occlusion consistency. The occlusion between sprites regulates the visible area of each sprite with

$$\mathcal{L}_{\text{occ}} = \begin{cases} ||1 - (V_{i,p})_\alpha||_2^2 & \text{if} \quad ||I_p - (\Omega_{i,p})_{\text{RGB}}||_2^2 \geq \mu_{\text{occ}}, \\ 0 & \text{others.} \end{cases} \quad (11)$$

where $||I_p - (\Omega_{i,p})_{\text{RGB}}||_2$ is the color distance between the original video and the composition of all sprites excluding the i-th sprite,*i.e.*, the color difference between compositions with and without the i-th sprite. If this difference is large enough (greater than a parameter μ_{occ}) at a position p, then this screen position must be occluded by the i-th sprite and the $(V_{i,p})_\alpha$ is encouraged to be large.

Optical flow consistency. Although our self-supervised framework can decompose sprites without any external data, we found training with optical flows can improve the separation of multiple sprites. As we discussed before, cartoon animation contains arbitrarily drawn motions of great complexity, making the pixel-level correspondence not always accurate. Our approach only *partially* trusts the optical flow. We decompose an input optical flow $F^{t1\to t2} \in \mathbb{R}^{w\times h\times 2}$ into a flow outside sprites $F_o^{t1\to t2} \in \mathbb{R}^{w\times h\times 2}$ and a flow inside sprites $F_i^{t1\to t2} \in \mathbb{R}^{w\times h\times 2}$ so as to establish the connection between the flow and sprites (fig. 6). The flow outside sprites is computed with

$$F_{o,p}^{t_1\to t2} = \sum_i (V_{i,\alpha})_p (H_{i,t2}^{-1} \cdot H_{i,t1} \cdot p - p) \quad (12)$$

where $H_{i,t2}^{-1} \cdot H_{i,t1} \cdot p$ means that we first transform a pixel position p from screen space to the sprite space at time t1 using $H_{i,t1}$, and then at time t2, we transform the position back to screen space using $H_{i,t2}^{-1}$. The offset $(H_{i,t2}^{-1} \cdot H_{i,t1} \cdot p - p)$ can be viewed as a flow outside the i-th sprite. By composing these offsets with the sprite visible areas $V_{i,\alpha}$, we obtains the outside flow F_o. The inside flow F_i an be written as a residual flow between the input flow and the outside flow

$$F_{i,p}^{t1\to t2} = F_p^{t1\to t2} - F_{o,p}^{t1\to t2} \quad (13)$$

and we regulate the inside flow with

$$\mathcal{L}_{\text{flow}} = \begin{cases} ||F_{i,p}^{t1\to t2}||_2^2 & \text{if} \quad ||F_{i,p}^{t1\to t2}||_2 \leq \mu_{\text{flow}}, \\ 0 & \text{others.} \end{cases} \quad (14)$$

where the inside flow F_i is minimized to zero if the original flow magnitude is relatively low. This makes sure that the dominant part of the motion is contributed by the outside transformations. We would like to point out that this regulation only partially trust the optical flow; only pixels with relatively low ($\leq \mu_{\text{flow}}$) inside flow are involved in the optimization. We set μ_{flow} as a very large number at the beginning of learning and then gradually reduce it during iterations.

3.3 Training

For each sprite, the trainable variables are a content matrix $C_i \in \mathbb{R}^{T\times w_i\times h_i\times 4}$, a transform matrix $H_i \in \mathbb{R}^{T\times 3\times 3}$, and a Pixel MLP M_i. Given any pixel position p, we use the sampling function $\phi(\cdot,\cdot)$ to obtain the content values as $\phi(C_i, p)$. We apply several schedules to make the training practical:

Automatic padding. When $\phi(\cdot,\cdot)$ receives a pixel position p outside the queried matrix (or when the p contains negative values), the ϕ automatically pad the matrix. The padding is limited to a maximum padding distance d_{max} to avoid memory leak.

Continuous sampling. To avoid aliasing caused by discrete sampling in both forward feeding and backward gradient propagation, we apply mipmap-like pyramid technique [Williams 1983] to each sprite. For each content matrix C_i, we generate at most 5 mipmap-like pyramid level $\texttt{mip}_{1...5}(C_i)$. We define $\texttt{mip}_0(C_i) = C_i$. Each map is independently trainable. The final sampling can be written as

$$\phi(C_i, p) = \sum_{k=0}^{5} \phi_b(\texttt{mip}_k(C_i), \frac{1}{2^k} p) \quad (15)$$

where $\phi_b(\cdot)$ is bilinear interpolation. After training, all maps are merged into original matrices using the same Eq. (15).

Valid transformation. We make sure that all the homogeneous transforms are valid by checking their determinants after each training iterations. If we find any transform matrix with zero determinant, we replace that matrix with a 3×3 identity matrix.

Pixel Multilayer Perceptron. The MLP contains 1 input layer (10 nodes), 8 fully-connected hidden layers (256 channels), and 1 output layer (4 nodes). All hidden layers are activated with Rectified Linear Unit (ReLU). The output layer is activated with Sigmoid. We do not use positional encoding.

3.4 Implementation Details

Our framework supports flexible resolutions and in most experiments our test animations are between 1280×720 and 256×128

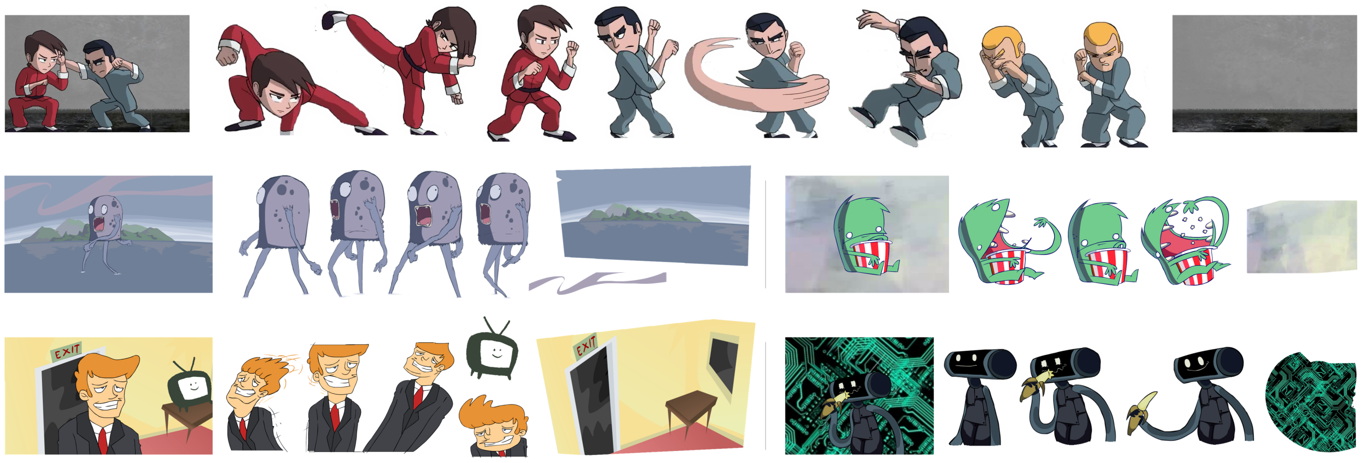


Fig. 7. **Qualitative results:** We present qualitative results of our approach. In each example, the left most image is from the input video, while all other images are from decomposed sprites. ©*Kung Fu, Water Monster, Popcorn, Office man, Robot and Banana, used with artist permission.*

with about 100 frames. All RGBA values are in the range between 0 and 1. The default parameters are $\alpha_{\text{rec}} = 1.0, \alpha_{\text{mem}} = 1.0, \alpha_{\text{occ}} = 1.0, \alpha_{\text{tv}} = 1.0, \alpha_{\text{flow}} = 1.0, \alpha_{\text{color}} = 1.0, \alpha_{\text{alpha}} = 1.0, \mu_{\text{occ}} = 0.1$, and μ_{flow} is linearly deceased from 100 to 0.1 during training. We train with 100K iterations each video. In the first 10K iterations, we set $\alpha_{\text{flow}} = 100.0$ as a warmup (we have 2 warmup stages: one for the alpha learning for the optical flow, and one for the initialization of the homogeneous transforms). The input optical flow is RAFT [Teed and Deng 2020]. We use a batch size of 10K pixel points. All sprite content matrices are initialized as the same size as the input video. The max distance of the automatic padding is $d_{\text{max}} = 512$. We use Adam optimizer and a learning rate of $1e - 4$. All sprite transforms are initialized with identity matrices. After training, we remove empty border areas (with zero opacity) in content matrices to obtain the sprite size $w_i \times h_i$. Training one animation video takes about 4 hours in a NVIDIA RTX 3080.

4 EXPERIMENT

4.1 Qualitative results

We present qualitative results of decomposed sprites in figure 7. The contents of the input animation covers a variety of topics, including human, animal, robots, *etc*. More results are presented in the video and supplementary materials.

4.2 Comparison to previous methods

We present a visual comparison of the decomposed sprites. We focus on the comparison to cartoon decomposition and video decomposition methods: (1) *Globally Optimal Toon Tracking* [Zhu et al. 2016] is a cartoon animation tracking method that detects the motion regions that shares high similarity. This method is a typical traditional method that depends on the cartoon edge and region extraction to process animations. This method does not by default merge regions into sprites, We achieve sprite decomposition using the connectivity detection in their framework [Zhu et al. 2016]. (2) *Omnimatte* [Lu et al. 2021] train neural networks to decompose any video into layers. This method is self-supervised and does not depend on prior knowledge of natural videos so that the method can also be applied to cartoon videos. (3) *Layered Neural Atlases* [Kasten et al. 2021] is a typical neural network method that decompose videos by fitting deformed 2D textures. This method models the animations in a video as rigid or non-rigid deformations of global textures. This allows the video to be decomposed in a self-supervised way. (4) *Deformable Sprites* [Ye et al. 2022] is an enhanced approach that also uses deformation model but can automatically determine the object masks (using optical flow guidance). This method is one of the state-of-the-art methods in video decomposition. It is worth noticing that the methods (2) and (3) may require users to give an initial coarse mask of the object for warmup stages. In this case, we use the automatically generated mask of (4), since the architecture of (3) and (4) shares similarity.

As shown in figure 8, we can see that traditional cartoon processing method [Zhu et al. 2016] may achieve sharp and clean results since the decomposition is based on image segmentation; the drawback is that the artifacts of mis-segmentation is non-trivial, and the occluded areas needs additional considerations. The deformation-based neural network approaches [Ye et al. 2022] and [Kasten et al. 2021] causes blurring artifacts, which is mainly caused by the global deformation formulation. Since cartoon animations contains many large and irregular motions, the pixel-space deformation is not sufficient to model all animations. [Lu et al. 2021] shows relatively better results because this method consider each frame independently and do not use global deformation constraints; the blurring artifacts of this method is caused by the appearance approximation of neural networks that can be ill-conditioned since a optimal global approximation can be undetermined. The sharp and clean result of our approach is mainly because of our "sprite-from-sprite" architecture that use some relatively simple sprites to guide the decomposition of more challenging sprites.

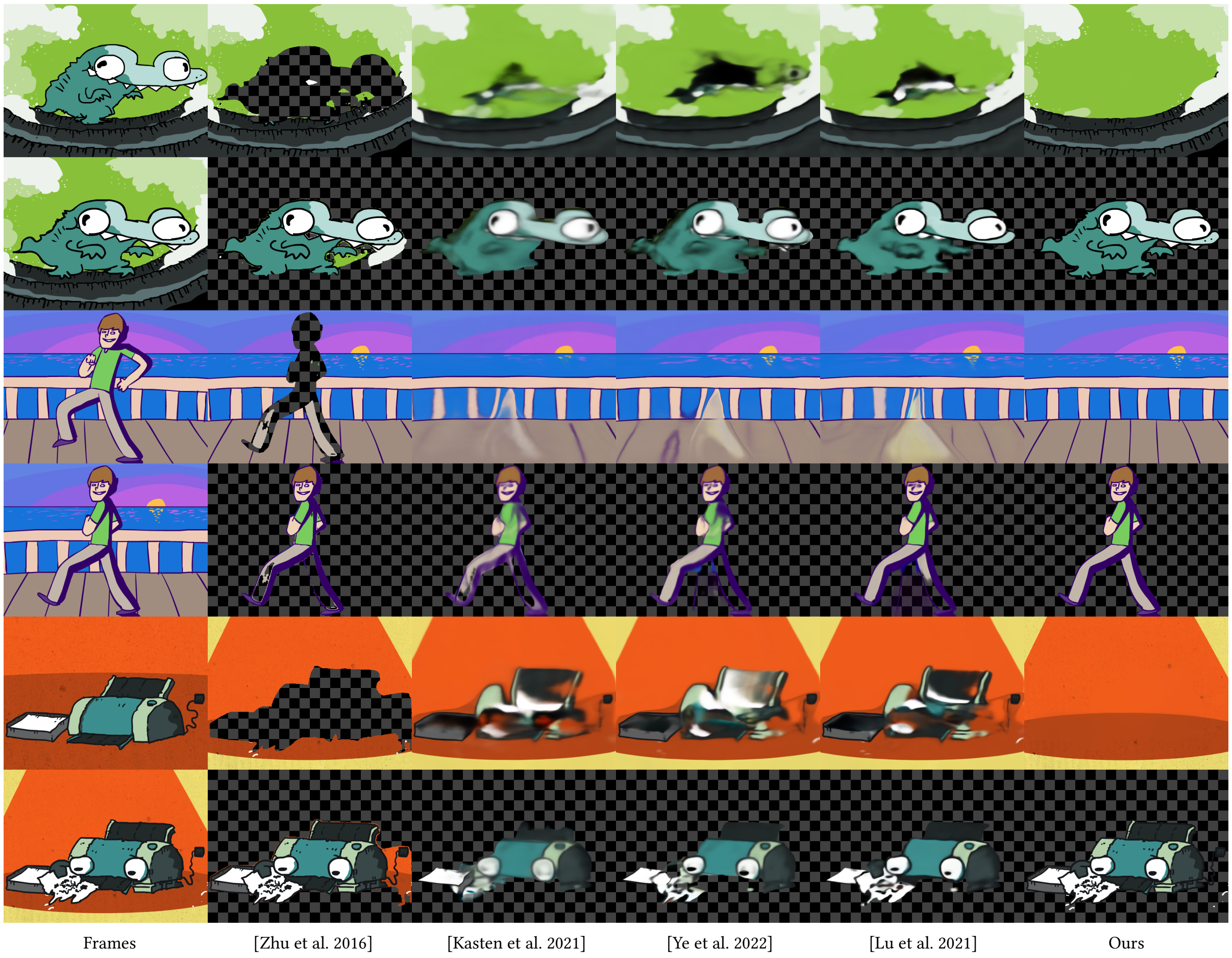


Fig. 8. **Comparison to previous methods:** We compare our result to previous related methods. The left most images are original frames from input animations. ©*Nerdy Dragon, Walk with Pride, Cute Printer, used with artist permission.*

4.3 Evaluation of segmentation masks

We compare the quality of our sprite masks to other unsupervised motion segmentation baselines using DAVIS2016 [Perazzi et al. 2016], FBMS [Ochs et al. 2014b], and Seg TrackV2 [Li et al. 2013]. We test with baseline methods [Koh and Kim 2017; Lao and Sundaramoorthi 2018; Yang et al. 2021b,a, 2019]. We also compare with more recent methods like [Ye et al. 2022]. We report quantitative performance in Table 1. We also include more results in Figure 9. Although our approach is not designed for video object segmentation, the overall performance is comparable with latest methods, *e.g.*, Video Sprites [Ye et al. 2022].

4.4 Perceptual user study

In order to perceptually evaluate and compare our approach with existing methods, we perform a perceptual user study focusing on human aspects of different decomposition algorithms. In particular, the user study involves 15 individuals, where 10 individuals are non-artist students, and the other 5 are professional artists. We sample 20 animation clips, and then use 5 methods to generate decompositions for each animation clip. This leads to 20 decomposition groups, with each group containing 5 results from 5 methods. The participants are invited to rank the results in each group. When ranking the results in each group, we ask users the question – "Which of the following results do you prefer most? Please rank the following sprites according to your preference". We use the Average Human Ranking (AHR) as the testing metric. For each group in the 150 groups, one random user ranks the 5 results in the current group from 1 to 5 (lower is better). Afterwards, we calculate the average ranking obtained by each method. We also report how many times a method is preferred by most of the users.

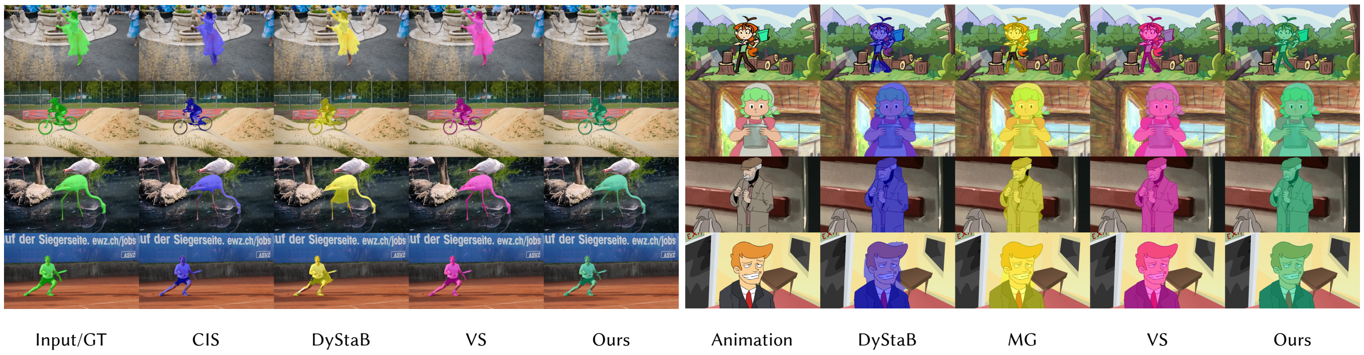


Fig. 9. **Results of mask segmentation:** We present mask segmentation results of our approach and related methods. The ground truth (GT) of real-world images are from the DAVIS dataset.*Cutting Tree, Fairy Tail, Old Man, Office Man, used with artist permission.*

Table 1. **Quantitative mask evaluation on VOS benchmarks.** We compare masks from our method with top-performing baselines on the video object segmentation datasets. The experiment setting is consistent with Deformable Sprites [Ye et al. 2022].

	DAVIS	FBMS	SegTV2
ARP [Koh and Kim 2017]	76.1	59.5	57.0
ELM [Lao and Sundaramoorthi 2018]	61.4	61.0	-
MG [Yang et al. 2021b]	68.0	50.5	57.9
CIS [Yang et al. 2019]	71.2	63.5	62.3
DyStaB [Yang et al. 2021a]	80.5	71.2	74.3
DF [Ye et al. 2022]	79.2	71.8	72.3
Ours	78.2	71.9	70.1

Table 2. **Average Interaction Time and User Ranking.** We report the recorded preferred case counts and user ranking (1 to 5 indicates worst to best) of different methods. For the preferred cases, we show both the integer value of how many times a method is preferred by most users, and percentage value of the soft user preferences. 15 users are involved in the user study. Best results are shown in bold.

Method	Preferred cases	Avg. rank
[Zhu et al. 2016]	0 (0.1%)	2.9 ± 0.5
[Ye et al. 2022]	0 (0.1%)	1.8 ± 0.3
[Kasten et al. 2021]	0 (0.1%)	1.7 ± 0.2
[Lu et al. 2021]	1 (4.6%)	3.9 ± 0.6
Ours	**19** (95.1%)	**4.6** ± **0.1**

We compare our approach with [Zhu et al. 2016], [Ye et al. 2022], [Kasten et al. 2021], and [Lu et al. 2021] Results are shown in Table 2. We find that users prefer our approach over all other approaches (in 19/20 cases). We also observe that: (1) Our framework outperforms the secondly best method by a large margin of 0.7/5. (2) The method [Lu et al. 2021] reports the secondly best score. (3) The two deformation-based methods [Kasten et al. 2021; Ye et al. 2022] reports similar perceptual quality, with [Ye et al. 2022] slightly better than [Kasten et al. 2021]. (4) The traditional approach [Zhu et al. 2016] reports a relatively medium performance, which is mainly because image segmentation method can also produce sharp and clean results but the mis-segmentation may reduce the user preference rate.

4.5 Ablative study

We present an ablative study in figure 10. We first focus on an alternative architecture where we remove the "sprite-from-sprite" learning by simply set all Ω_i as all zero matrices. This will directly disconnect the sprite subset composition step and the main MLP. No that even in this configuration, the MLP can still learns a mapping, to some extent, as the MLP still receives the entire input animation video and can learn to segment the video only using the input x,y,t index and RGB values. We present examples in figure 10a, where we can see that without the guidance of those relatively simple sprites, the decomposition of the challenging sprites is difficult and can fail the model into collapse modes.

We also discuss the effect of the occlusion consistency. Without the occlusion consistency (figure 10c), the model may cheat to learn a color palette and then use color blending to reconstruct the original animation. A learning objective of occlusion consistency can effectively reduce these artifacts. Furthermore, if our framework is trained without optical flows, the separation capability of multiple objects becomes a bit weaker. For example the hand of the boy (figure 10c) may have "ghost" artifacts in the background sprite if trained without optical flow consistency. Finally, the proposed full method (figure 10d) can facilitate the separation of multiple sprites by achieving a balance between these consistencies.

4.6 Applications

With the decomposed sprites available, we can achieve a variety of manipulations of the input cartoon animation. As shown in figure 11, we can see that a re-composing of the animation can alter the ratio of cartoons so that it can be displayed onto screen devices with difference ratios. We also demonstrate that the decomposed cartoon sprites can also be used in recoloring, retiming, or reposing the animations, *e.g.*, the animation of the two robots in figure 12.

To be specific, in order to change the transform (or timing) of a sprite, we apply a global offset (or bilinear resampling, respectively) on the transformation sequence $\boldsymbol{H}_i$. The recoloring is also a global

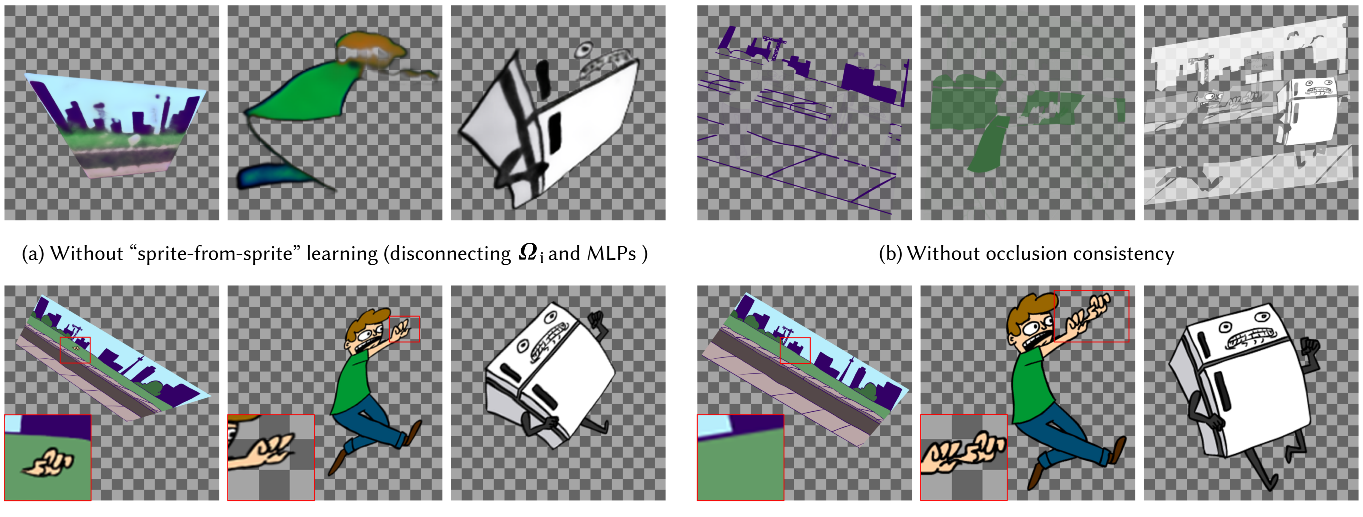

(a) Without "sprite-from-sprite" learning (disconnecting Ω_i and MLPs)
(b) Without occlusion consistency
(c) Without optical flow consistency
(d) Full method (proposed)

Fig. 10. **Ablative study:** We investigate the influence of each components in our framework by removing them one-by-one. We present sprites with the most salient differences. ©*Sandwich Town, used with artist permission.*

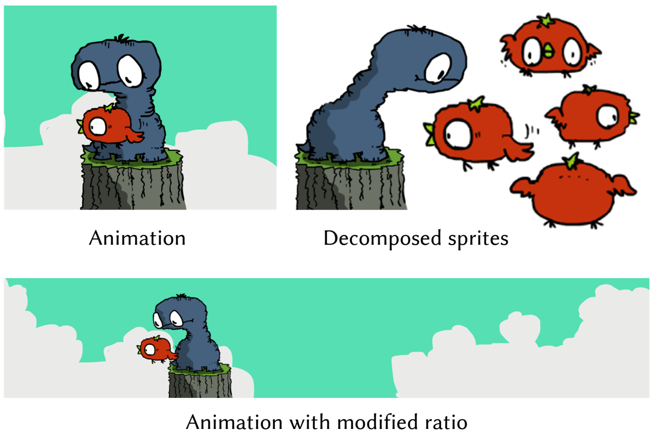


Fig. 11. **Adjusting animation ratio:** By decomposing and composing sprites, we can change the ratio of an animation. In this example, we present an extreme case where the width is significantly enlarged. ©*Strange Bird, used with artist permission.*

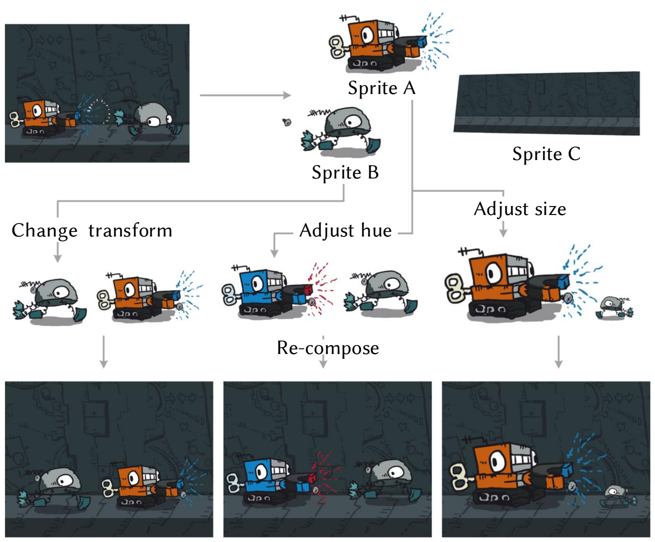


Fig. 12. **Frame content manipulation:** All the adjustment are global manipulations and all frames of one sprite only need one editing. By decomposing animation into sprites, we can achieve various manipulations like recoloring, reposing, retiming, *etc.* ©*Chasing the Robot, used with artist permission.*

editing, we recolor a sprite with

$$f(c) = ac + (1 - a)y \quad \text{where} \quad a = \text{clamp}(||c - x||_1) \tag{16}$$

where $f(\cdot)$ is a recoloring mapping applied to all pixels in one sprite. The c is pixel color, x is color to change (the original color), and y is the target color (the new color). The operation "clamp" will clip any value to the range [0,1].

4.7 Comparison to real sprites

We were also interested in the question of how close our generated sprites were to the artist's real sprites. We invited the artist to give us the real composition sprites and then we compared our generated results with the artist's data. As shown in figure 13, we can see that some parts of our extracted sprites are consistent with the artist's real sprites, *e.g.*, the bird that is animated in a separated sprite. But we also notice that the real sprites are often much more detailed, *e.g.*, the windows and wall paintings above the room in the background are artist-defined sprites. Automatically extracting these sprites can be very difficult because these elements are relatively stationary in

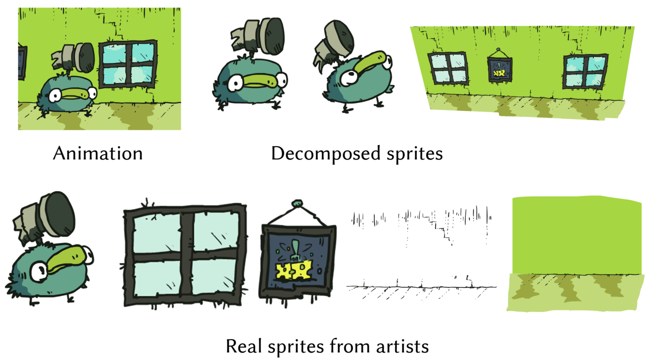


Fig. 13. **Comparison to real sprites:** We present a comparison between our decomposed sprites and ground truth sprites given by one professional artist. ©*Hammerbird, used with artist permission.*

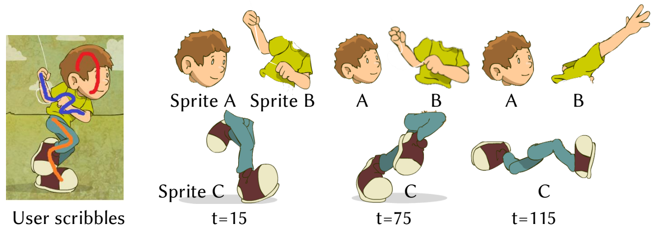


Fig. 14. **Interactive decomposition with user scribbles:** Our framework can receive user scribbles to facilitate richer decomposition,*e.g.*, extracting more layers inside sprites. In this example, the user scribble annotate one single frame. ©*Boy flying a kite, used with artist permission.*

motion and no motion cues can be used, so new prior knowledge, assumptions of colors, or user interaction need to be introduced for further decomposition.

4.8 Interactive decomposition with user scribbles

When users want to guide the decomposition, they can use scribbles to label some areas to make the distribution of sprites more suitable for their actual needs. For example, in figure 14, the pixels covered by scribbles are considered to be labeled with categories. We directly use Mean Squared Error (MAE) to make the local transparency (the alpha channel of $\boldsymbol{V}_i$) of the corresponding sprites on these pixels as close to "1" as possible. We can see that such interaction can support a more detailed decomposition that better meets the specific needs of artists in their daily work.

To be specific, we optimize an additional loss with

$$\mathcal{L}_{\text{user}} = \boldsymbol{U}_i||\boldsymbol{V}_i - 1||_2^2 \quad (17)$$

where $\boldsymbol{U}_i$ is the user-specified binarized masks for each sprites. To facilitate straight forward editing, the user interface can use scribbles with different colors, and then convert colored scribbles to multiple masks.

4.9 Comparison in different applications

We compare our approach to MarioNette [Smirnov et al. 2021] and DTI-Sprites [Monnier et al. 2021] in their experimental settings. MarioNette is inspired from screen capture of a video game like Mario. This experiment is targeted at extracting sprite elements from gaming videos. We use a typical experimental setting of four moving gameobjects. The results are presented in figure 15. We can see that this approach achieves comparable results to MarioNette. DTI-Sprites is another neural representation methods that can decompose videos in a self-supervised manner. We also attach the comparison to DTI-Sprites.

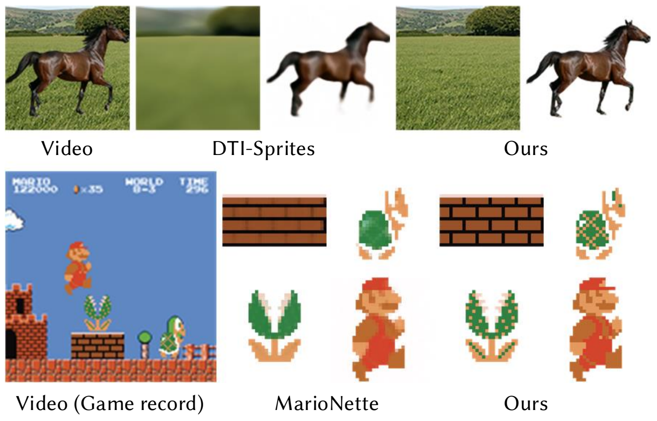


Fig. 15. **Comparison in other applications:** We compare our method to MarioNette and DTI-Sprites. We use the experimental settings as recommended in their papers.

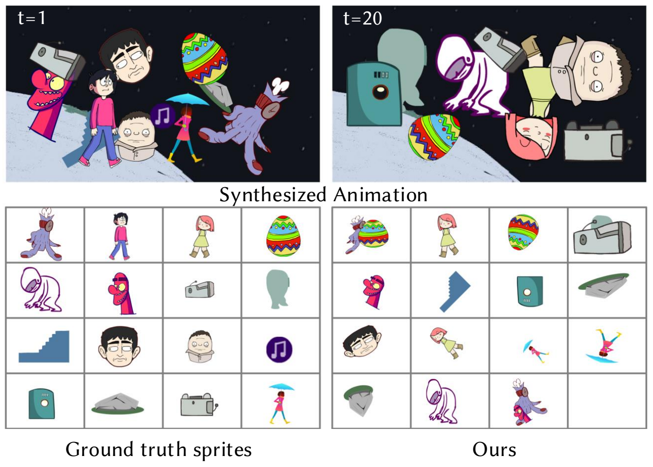


Fig. 16. **Experiment on synthetic sprites:** We test our method with relatively large amount of sprites at relatively low resolution.

4.10 Robustness and Noise tolerance

We perform a robustness test using a synthetic experimental setting. We synthesize animation with relatively more sprites at relatively low resolutions and test the capability of framework. To be specific, we synthesize animations using one background and 16 foreground random moving sprites. All sprites are real sprites provided by artists. The moving include translating and rotating. The results are

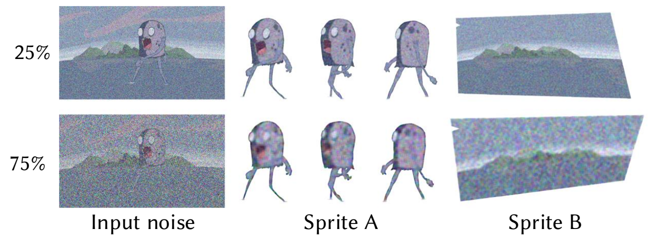


Fig. 17. **Noise tolerance test:** We test our approach with different synthesized noise levels to test the framework robustness. The noise are Gaussian distribution in RGB place that replaces the pixel colors at a certain percentage as shown in the figure.©*Water Monster, used with artist permission.*

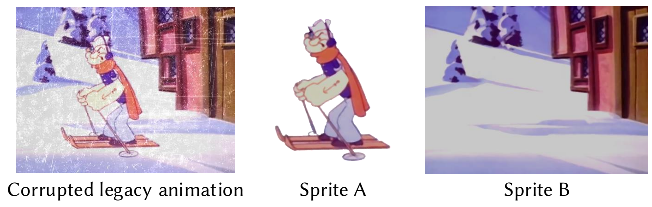


Fig. 18. **Handling corrupted legacy animation:** We test our approach using legacy animation videos with severe film noise.

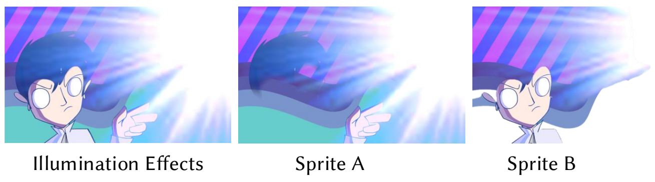


Fig. 19. **Videos with Illumination effects (failure case):** We test our approach using animations with strong illumination effects with flashy light and shadows. ©*Flash boy, used with artist permission.*

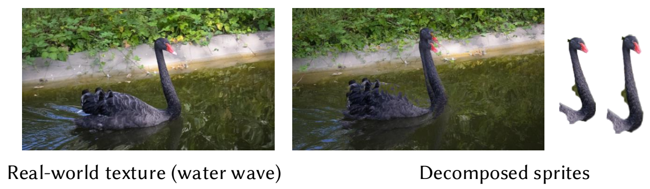


Fig. 20. **Videos with real-world texture (failure case):** In this example, the wave of water is a global animated texture throughout the video. The video is from DAVIS dataset.

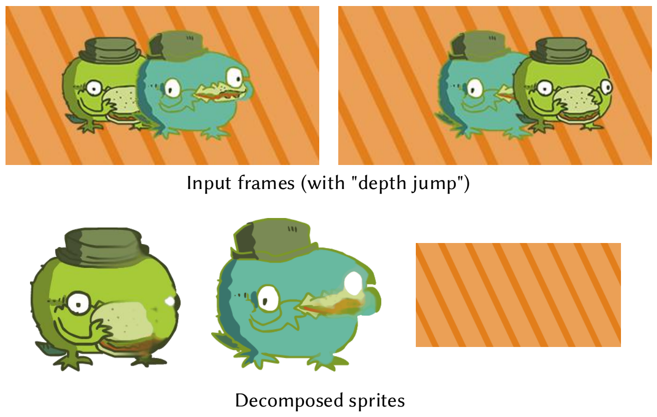


Fig. 21. **Videos with "depth jump" (failure case):** We test our approach using videos with sprites that "jumps" between depth layers. ©*Birds and Burgers, used with artist permission.*

presented in figure 16. We can see that, although not perfect, this method can extract visually salient sprites from the animation.

We also test the method with noise tolerance. When the pixel noise of standard normal distribution is applied to the videos, the quality of sprites degrade, while the boundary of the sprites are still sharp and clean, as shown in Figure 17. We also test with corrupted animations from legacy footage as shown in Figure 18. Our discovery is that the “legacy footage” (or film noise) can be smoothed by this method.

4.11 Limitation

First, since our method utilizes the color appearance of the animation, the MLP may not have a way to get a valid input if the animation does not have continuous color, *e.g.*, line drawing animations. Second, when complex lighting changes inside the scene, even if the sprites are successfully decomposed, the lighting will still remain on the sprite’s appearance instead of being decomposed out. This may require further intrinsic decomposition or the addition of additional advanced blending layers that can be learned. Finally, it can be difficult to separate multiple objects if they are completely stationary and without any motion differences. We also note several additional failure cases:

Intensive illumination effects. As shown in Figure 19, when the input animation has strong illumination effects like flashing and noisy shadow, the recognition of sprites can be influenced, leading to inaccurate sprites extractions.

Real-world animated texture on objects. As shown in Figure 20, when objects contains animated texture, *e.g.*, the wave on the water is animated throughout the video, the texture animation can lead to defective sprite alignment and then fail the method to decompose usable sprites.

Sprites with “depth” jumps. As shown in Figure 20, since all sprites are optimized in a fixed “layer sequence”, if the depth order/layer of objects changes in the video, the quality of sprites can be influenced.

5 CONCLUSION

We have proposed a MLP-based neural network framework to decompose “sprites” from cartoon animations. The method is based on the observation that cartoon animations may consist of both simple and challenging sprites: the simple sprites may have regular motion while the challenging sprites may have complicated animations and hand-drawn arbitrary motions. Our framework can automatically discover sprites with simple and regular motion, and at the same time, use those simple sprites to guide the decomposition of more complicated sprites. This approach is named after "sprite-from-sprite" decomposition. We hope this method lead to sharp-and-clean decompositions can be applied to many artistic use cases.

ACKNOWLEDGMENTS

This project is supported by the Research Grants Council of the Hong Kong Special Administrative Region, under RGC General Research Fund (Project No. CUHK14201017 and CUHK14201918). This work is also supported by Style2Paints Research — Interactive Research, Real Demand, and Constructive Feedback. We thank all reviewers and referee members for evaluating and improving this work. We also thank all artists for creating and providing the involved artworks.

REFERENCES

Yagiz Aksoy, Tunç Ozan Aydin, Aljosa Smolic, and Marc Pollefeys. 2017. Unmixingbased soft color segmentation for image manipulation. *ACM Transactions on Graphics* (2017).

Thomas Brox and Jitendra Malik. 2010. Object Segmentation by Long Term Analysis of Point Trajectories.

Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. 2015. Palette-based Photo Recoloring. *ACM Transactions on Graphics* (2015).

Achal Dave, Pavel Tokmakov, and Deva Ramanan. 2019. Towards Segmenting Anything That Moves. In *ICCV Workshops*.

Marek Dvorožňák, Wilmot Li, Vladimir G. Kim, and Daniel Sýkora. 2018. ToonSynth: Example-Based Synthesis of Hand-Colored Cartoon Animations. *ACM Transactions on Graphics* 37, 4, Article 167 (2018).

Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. 2018. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 216–224.

Ondřej Jamriška, Šárka Sochorová, Ondřej Texler, Michal Lukáč, Jakub Fišer, Jingwan Lu, Eli Shechtman, and Daniel Sýkora. 2019. Stylizing Video by Example. *ACM Transactions on Graphics* 38, 4, Article 107 (2019).

Yoni Kasten, Dolev Ofri, Oliver Wang, and Tali Dekel. 2021. Layered Neural Atlases for Consistent Video Editing.

Margret Keuper. 2017. Higher-Order Minimum Cost Lifted Multicuts for Motion Segmentation.

Margret Keuper, Bjoern Andres, and Thomas Brox. 2015. Motion Trajectory Segmentation via Minimum Cost Multicuts.

Yeong Jun Koh and Chang-Su Kim. 2017. Primary Object Segmentation in Videos Based on Region Augmentation and Reduction.

Dong Lao and Ganesh Sundaramoorthi. 2018. Extending layered models to 3d motion. In *Proceedings of the European conference on computer vision (ECCV)*. 435–451.

Fuxin Li, Taeyoung Kim, Ahmad Humayun, David Tsai, and James M Rehg. 2013. Video segmentation by tracking many figure-ground segments.

Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. 2020. Neural Scene Flow Fields for Space-Time View Synthesis of Dynamic Scenes. *arXiv preprint arXiv:2011.13084* (2020).

T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. 2017. Focal loss for dense object detection. *ICCV* (2017).

Xueting Liu, Xiangyu Mao, Xuan Yang, Linling Zhang, and Tien-Tsin Wong. 2013. Stereoscopizing Cel Animations. *ACM Transactions on Graphics (SIGGRAPH Asia 2013 issue)* 32, 6 (November 2013), 223:1–223:10.

Erika Lu, Forrester Cole, Tali Dekel, Weidi Xie, Andrew Zisserman, David Salesin, William T Freeman, and Michael Rubinstein. 2020. Layered Neural Rendering for Retiming People in Video. In *SIGGRAPH Asia*.

Erika Lu, Forrester Cole, Tali Dekel, Andrew Zisserman, William T Freeman, and Michael Rubinstein. 2021. Omnimatte: Associating Objects and Their Effects in Video. In *CVPR*.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4460–4470.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*. Springer, 405–421.

Tom Monnier, Elliot Vincent, Jean Ponce, and Mathieu Aubry. 2021. Unsupervised Layered Image Decomposition into Object Prototypes. In *ICCV*.

R.M.H. Nguyen, S. Cohen B. Price, and M. S. Brown. 2017. GroupTheme Recoloring for Multi-Image Color Consistency. *Computer Graphics Forum* (2017).

Peter Ochs and Thomas Brox. 2011. Object segmentation in video: A hierarchical variational approach for turning point trajectories into dense regions.

Peter Ochs, Jitendra Malik, and Thomas Brox. 2014a. Segmentation of moving objects by long term video analysis. 36, 6 (2014).

P. Ochs, J. Malik, and T. Brox. 2014b. Segmentation of moving objects by long term video analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 6 (Jun 2014), 1187 – 1200. Preprint.

Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. 2021. Neural Scene Graphs for Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2856–2865.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. 2016. A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation.

Alex Rav-Acha, Pushmeet Kohli, Carsten Rother, and Andrew Fitzgibbon. 2008. Unwrap Mosaics: A New Representation for Video Editing . In *SIGGRAPH '08 ACM SIGGRAPH 2008 papers*.

Jianbo Shi and J. Malik. 1998. Motion segmentation and tracking using normalized cuts.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems* 33 (2020).

Dmitriy Smirnov, Michael Gharbi, Matthew Fisher, Vitor Guizilini, Alexei A. Efros, and Justin Solomon. 2021. MarioNette: Self-Supervised Sprite Learning.

Daniel Sýkora, Mirela Ben-Chen, Martin Čadík, Brian Whited, and Maryann Simmons. 2011. TexToons: Practical Texture Mapping for Hand-drawn Cartoon Animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*. 75–83.

Daniel Sýkora, Jan Buriánek, and Jiří Žára. 2005. Sketching Cartoons by Example. In *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling*. 27–34.

Jianchao Tan, Jyh-Ming Lien, and Yotam Gingold. 2017. Decomposing Images into Layers via RGB-space Geometry. *ACM Transactions on Graphics* (2017).

Jianchao Tan, Jyh-Ming Lien, and Yotam Gingold. 2018. Efficient palette-based decomposition and recoloring of images via RGBXY-space geometry. *ACM Transactions on Graphics* (2018).

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. *NeurIPS* (2020).

Zachary Teed and Jia Deng. 2020. RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. arXiv:2003.12039 [cs.CV]

Lance Williams. 1983. Pyramidal Parametrics.

Christopher Xie, Yu Xiang, Zaid Harchaoui, and Dieter Fox. 2019. Object Discovery in Videos as Foreground Motion Clustering.

Jun Xing, Li-Yi Wei, Takaaki Shiratori, and Koji Yatani. 2015. Autocomplete Hand-Drawn Animations. *ACM Trans. Graph.* 34, 6 (2015).

Charig Yang, Hala Lamdouar, Erika Lu, Andrew Zisserman, and Weidi Xie. 2021b. Self-supervised Video Object Segmentation by Motion Grouping.

Yanchao Yang, Brian Lai, and Stefano Soatto. 2021a. DyStaB: Unsupervised Object Segmentation via Dynamic-Static Bootstrapping.

Yanchao Yang, Antonio Loquercio, Davide Scaramuzza, and Stefano Soatto. 2019. Unsupervised Moving Object Detection via Contextual Information Separation.

Vickie Ye, Zhengqi Li, Richard Tucker, Angjoo Kanazawa, and Noah Snavely. 2022. Deformable Sprites for Unsupervised Video Decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hong-Xing Yu, Leonidas J. Guibas, and Jiajun Wu. 2022. Unsupervised Discovery of Object Radiance Fields.

Qing Zhang, Chunxia Xiao, Hanqiu Sun, and Feng Tang. 2017. Palette-Based Image Recoloring Using Color Decomposition Optimization. *IEEE Transactions on Image Processing* (2017).

Song-Hai Zhang, Tao Chen, Yi-Fei Zhang, Shi-Min Hu, and Ralph R. Martin. 2009. Vectorizing Cartoon Animations. *TVCG* (2009).

Haichao Zhu, Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. 2016. Globally Optimal Toon Tracking. *ACM Transactions on Graphics* 35, 4 (July 2016), 75:1–75:10.