Supplementary of Real-time Dynamic Texture Synthesis Using Neural Cellular Automata

Ehsan Pajouheshgar*, Yitao Xu*, Tong Zhang, Sabine Süsstrunk
School of Computer and Communication Sciences, EPFL, Switzerland
{ ehsan.pajouheshgar, yitao.xu, tong.zhang, sabine.susstrunk }@epfl.ch

Contents

1. User Study **1**

2. Demo **2**
2.1. Qualitative Results 2
2.2. Quantitative Results 2

3. Training Details of DyNCA **7**
3.1. Motion From Vector Field 7
3.1.1 Mathematical Expression of Target Vector Fields 7
3.2. Motion From Video 8
3.2.1 Dynamic Texture Synthesis 8
3.2.2 Choice of NCA Interval in Video Motion 8
3.2.3 Dynamic Style Transfer 8
3.3. Limitations 9

4. Quantitative Evaluation of Multi-scale Perception Ablation Study **9**

5. Ablation Study for Positional Encoding **9**
5.1. Motion from Vector Field 9
5.1.1 Qualitative Analysis 10
5.1.2 Quantitative Analysis 10
5.2. Motion from Video 11
5.2.1 Qualitative Analysis 11
5.2.2 Quantitative Analysis 11

6. More Results on Dynamic Texture Synthesis **12**

7. More Results on Dynamic Style Transfer **12**

1. User Study

We follow the user-study settings introduced by Tesfaldet et al. [4]. The participants see two videos in random order, one after another, and are asked to choose the video that appears more realistic. Tesfaldet et al. [4] in their study use 12-frame videos with 10 frames-per-second (fps), resulting in videos of 1.2 seconds in length. However, they loop these videos to create longer videos in order to study the effect of exposure time. Their results show that for exposure times ≥ 1.2 seconds, the accuracy of the participants in detecting the real video saturates and does not improve much. Moreover, they only compare the videos synthesized by their method with the real videos.

Our user study is slightly different from the one conducted by Tesfaldet et al. [4]. **First**, we limit ourselves to the original video length (1.2 seconds) to avoid looping artifacts. This choice is also justified by the results from [4] since there is not much difference in their results for exposure times ≥ 1.2 seconds. **Second**, in addition to Tesfaldet et al. [4], we experiment with videos synthesized by different SOTA methods to be able to compare their realism. We compare 4 different methods, including ours, with each other and with the real videos. These 4 methods are: DyNCA (Ours), (A) by Tesfaldet et al. [4], (B) Config FC of STGConvNet by [5], (C) Config ST of STGConvNet by [5].

We conduct our user study on the Amazon Mechanical Turk (AMT) platform. We use the same 59 dynamic texture videos provided by Tesfaldet et al. [4]. Each participant sees 59 pairs of videos. Similar to [4], the first 3 videos are warm-up comparisons and are not considered in the final results. As suggested in [4], we also chose 3 videos with very low quality, where the quality discrepancy is evident, as sentinel videos. We thus only consider the responses of participants who provide correct answers to those 3 sentinel videos. We then use the remaining 53 video comparisons in the evaluation.

We randomly create 100 different experiments by shuffling the order of 59 videos and also by shuffling the video pairs for each of the 59 videos. Each of the experiments can be done by a maximum of 3 different participants, and we do not allow the participants to participate more than once in our user study. Given the constraints above on the AMT platform, we received valid responses from 163 unique participants, which makes the total number of pairwise comparisons equal to $163 \times 53 = 8639$.

Table 1 shows the results. Each entry in the table indicates the number of times that the video of the corresponding column is chosen over the video of the corresponding

row as the more realistic video. The realism score presented in the last row shows the overall percentage that the corresponding column was chosen as the more realistic video. These results demonstrate that DyNCA has the highest realism score (54.9%) among the DyTS methods.

	Real	DyNCA	A [4]	B [5]	C [5]
Real	0	249	218	196	69
DyNCA	680	0	342	394	175
A [4]	625	520	0	413	231
B [5]	624	469	381	0	127
C [5]	819	695	680	732	0
Total Won	2748	1933	1621	1735	602
Total Lost	732	1591	1789	1601	2926
Realism Score	79.0% ±0.7%	54.9% ±0.8%	47.5% ±0.9%	52.0% ±0.9%	17.1% ±0.6%

Table 1. Pair-wise comparison results from our user study. The participants see two videos, one after another, in random order and are asked to choose the video that appears more realistic. Each entry in the table shows the number of times that the video of the corresponding column was chosen over the video of the corresponding row as the more realistic video. The realism score presented in the last row shows the overall percentage that the corresponding column was chosen as the more realistic video. Our DyNCA achieves the highest realism score compared to the other DyTS methods (54.9%).

2. Demo

Our demo is built on top of the open-source demo provided by Niklasson et al. [3] in their Self-Organizing Textures paper. Niklasson et al. implement the NCA model using Javascript and WebGL frameworks. We implement DyNCA by building on top of their codebase. Figure 2 and 3 show screenshots of our real-time interactive demo available at https://dynca.github.io. We also provide the synthesized videos for our Dynamic Style Transfer experiments at https://dynca.github.io/#style_transfer.

Besides the screenshots of our demo, we benchmark the performance of the WebGL demo on various devices, including computers and smartphones, and show that DyNCA can synthesize dynamic texture videos in real time on low-end GPUs.

2.1. Qualitative Results

Demo Type: Vector Field Motion: Figure 2 shows the demo of DyNCA models that are trained with vector field supervision. The users can choose the desired target vector field and target appearance, and our demo will synthesize the video in real time. The discussed real-time editing controls including *Speed Control*, *Direction Control*, *Editing Brush*, and *Local Coordinate Transformation* are also shown in Figure 2. The users can click on the canvas to edit the synthesized video using the brush tool. Our demo also allows the users to control the resolution by the resolution sliders. The resolution control simply changes the spatial dimensions of the seed H and W and re-runs the DyNCA model. The DyNCA models provided in *Vector Field Motion Demo* are trained without *Multi-Scale Perception* and with a seed size of 128×128. The users can also choose between the DyNCA-S and DyNCA-L configurations.

Demo Type: Video Motion: Figure 3 shows a screenshot of the DyNCA demo with models trained with video supervision. The users can choose the desired video used to train the DyNCA model. Here, we train the DyNCA models with *Multi-Scale Perception* enabled and a seed size of 256×256. The users can also choose between the DyNCA-S and DyNCA-L configurations.

2.2. Quantitative Results

For benchmarking the performance of our real-time DyNCA demo, we run the model for 500 steps to calculate the statistics of steps-per-second, frames-per-second (FPS), and how many milliseconds it takes for generating one frame. We show the results for various devices including computers and smartphones in Table 2 and 3. Table 2 summarizes the vector field motion performance when we map 24 NCA steps to one frame, and Table 3 the video motion when we map 64 NCA steps to one frame. The models trained for video motion are slightly slower because of utilizing multi-scale perception.

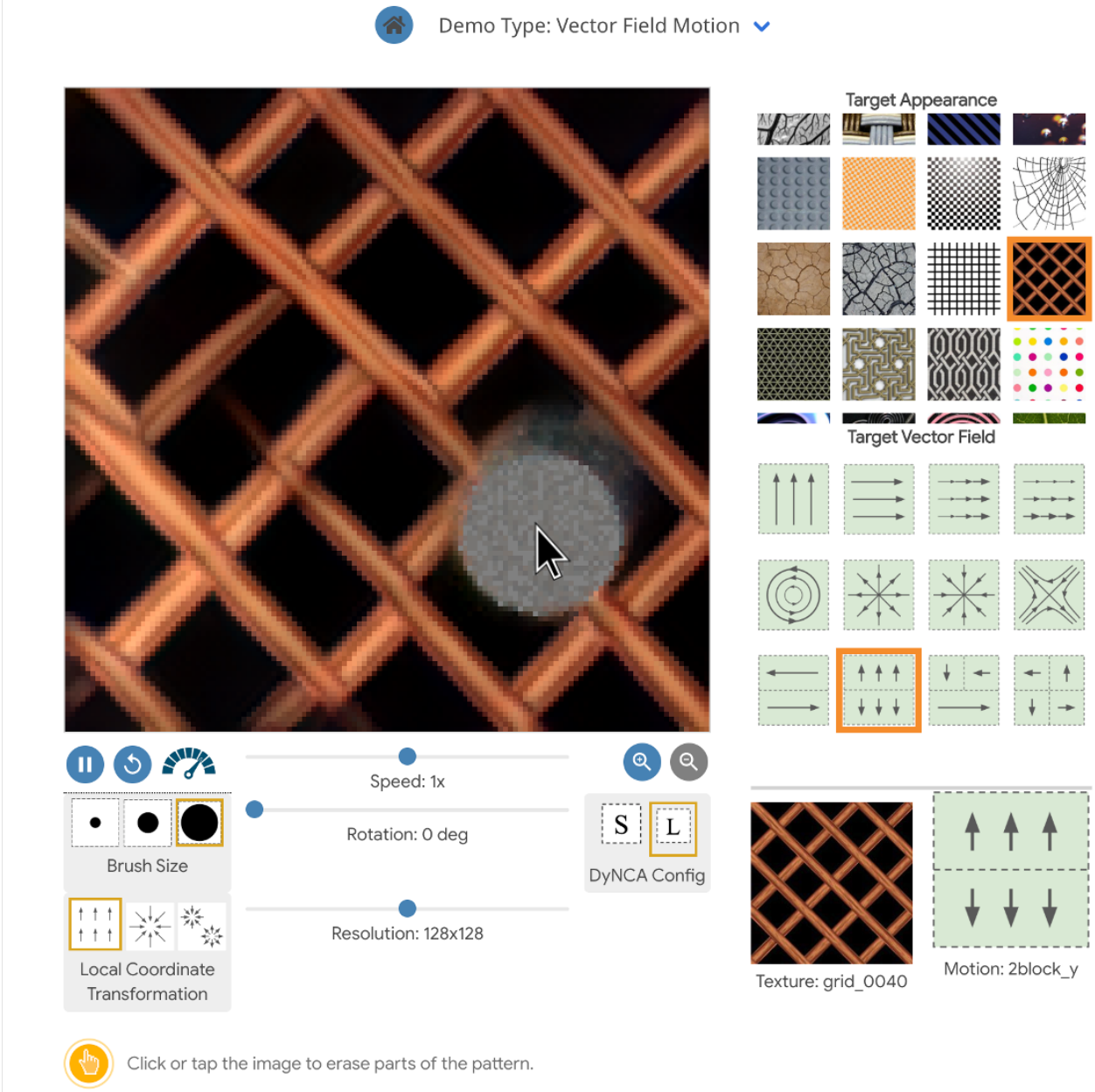

Figure 2. A screenshot for our demo on vector field motion. Users can choose from 45 different target appearance images and 12 different target vector fields, 2 different DyNCA configurations, and 3 different local coordinate transformations.

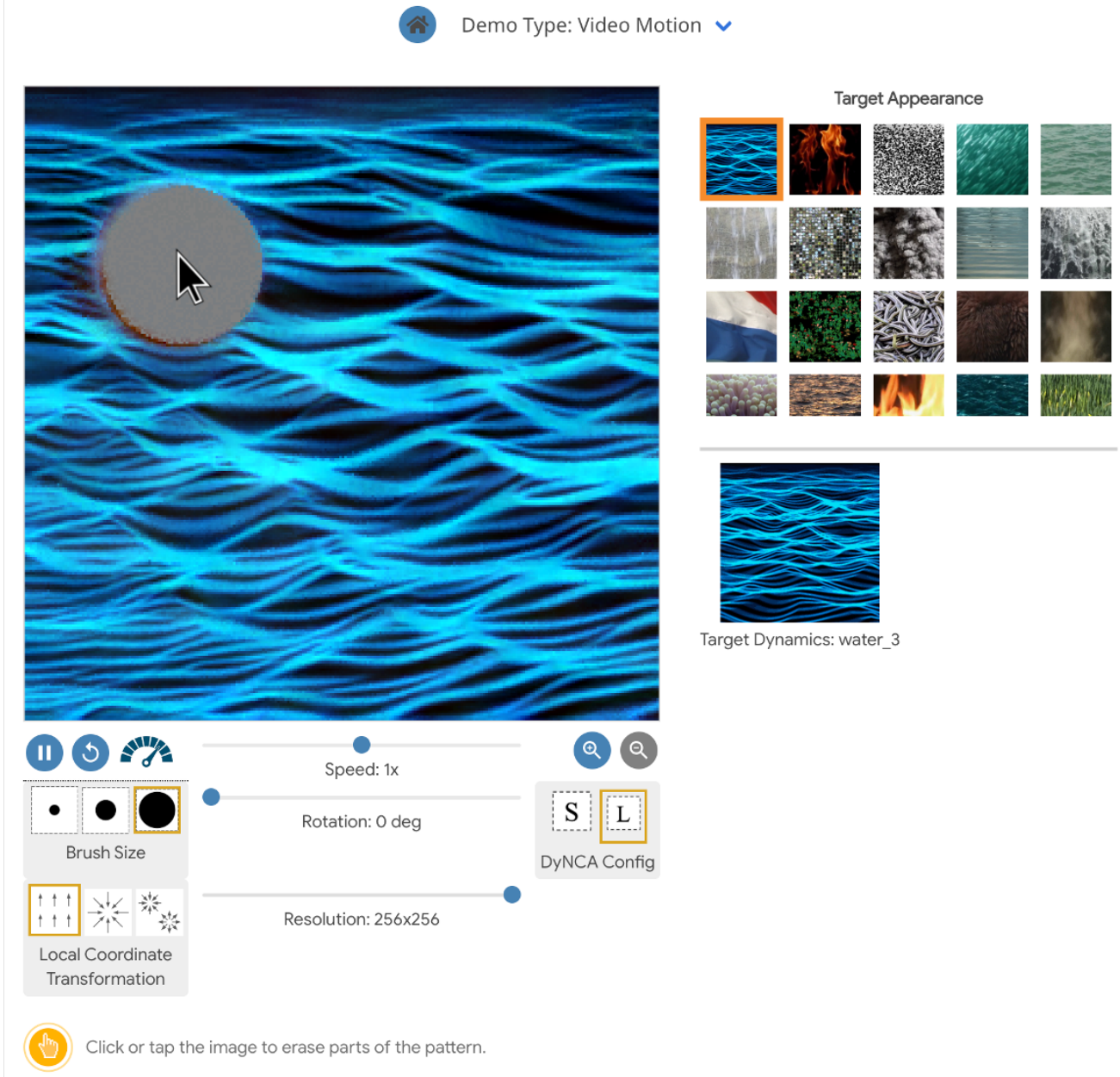

Figure 3. A screenshot for our demo on video motion. Users can choose from 59 different target videos, 2 different DyNCA configurations, and 3 different local coordinate transformations.

Device name-CPU-GPU	DyNCA Config	Seed Size	steps/s ↑	FPS ↑	ms/step ↓
Desktop Computer AMD Ryzen 3970X NVIDIA GeForce RTX 3090	S	128×128	14286	595.24	0.07
		256×256	4717	196.54	0.21
	L	128×128	10000	416.67	0.1
		256×256	2688	112.01	0.37
MacBook Pro (16-inch, 2021) Chip Apple M1 Pro	S	128×128	2747	114.47	0.36
		256×256	879	36.61	1.14
	L	128×128	1838	76.59	0.54
		256×256	569	23.70	1.76
Terrans Force X711 PLUS 67SH1 Intel(R) Core(TM) i7-6700K GTX 980M GPU	S	128×128	2212	92.18	0.45
		256×256	484	20.17	2.07
	L	128×128	1381	57.55	0.72
		256×256	303	12.64	3.30
MacBook Pro (13-inch, 2020) 2.3 GHz Quad-Core Intel Core i7 Intel Iris Plus Graphics 1536 MB	S	128×128	1299	54.11	0.77
		256×256	247	10.28	4.05
	L	128×128	818	34.10	1.22
		256×256	160	6.67	6.25
Apple iPhone 14 Hexa-core Apple GPU (5-core graphics)	S	128×128	1037	43.22	0.96
		256×256	217	9.06	4.60
	L	128×128	643	26.78	1.56
		256×256	130	5.42	7.69
Samsung Galaxy Z Flip4 Octa-core CPU Adreno 730 GPU	S	128×128	1029	42.87	0.97
		256×256	223	9.30	4.48
	L	128×128	608	25.34	1.64
		256×256	141	5.88	7.09
iPad Air (4th generation) A14 Bionic chip	S	128×128	697	29.06	1.43
		256×256	139	5.78	7.21
	L	128×128	450	18.75	2.22
		256×256	93	3.87	10.76
Apple iPhone 12 A14 Bionic chip	S	128×128	717	29.89	1.39
		256×256	162	6.74	6.18
	L	128×128	457	19.04	2.19
		256×256	101	4.23	9.84

Table 2. Benchmark of DyNCA WebGL implementations on different devices. This table shows the results for the Demo Type *Vector Field Motion*. In this configuration we set $T = 24$, i.e., we map 24 DyNCA steps to one video frame.

Device name-CPU-GPU	DyNCA Config	Seed Size	steps/s ↑	FPS ↑	ms/step ↓
Desktop Computer AMD Ryzen 3970X NVIDIA GeForce RTX 3090	S	128×128	10638	166.22	0.09
		256×256	2688	42.00	0.37
	L	128×128	8065	126.01	0.12
		256×256	2415	37.74	0.41
MacBook Pro (16-inch, 2021) Chip Apple M1 Pro	S	128×128	2326	35.34	0.43
		256×256	761	11.89	1.31
	L	128×128	1602	25.04	0.62
		256×256	508	7.93	1.97
Terrans Force X711 PLUS 67SH1 Intel(R) Core(TM) i7-6700K GTX 980M GPU	S	128×128	1923	30.05	0.52
		256×256	437	6.83	2.29
	L	128×128	1225	19.15	0.82
		256×256	278	4.35	3.59
MacBook Pro (13-inch, 2020) 2.3 GHz Quad-Core Intel Core i7 Intel Iris Plus Graphics 1536 MB	S	128×128	1055	16.48	0.95
		256×256	214	3.35	4.66
	L	128×128	682	10.66	1.47
		256×256	139	2.17	7.20
Apple iPhone 14 Hexa-core Apple GPU (5-core graphics)	S	128×128	929	14.52	1.08
		256×256	191	2.98	5.24
	L	128×128	597	9.32	1.68
		256×256	117	1.82	8.57
Samsung Galaxy Z Flip4 Octa-core CPU Adreno 730 GPU	S	128×128	873	13.63	1.15
		256×256	207	3.23	4.84
	L	128×128	557	3.96	3.94
		256×256	129	2.02	7.74
iPad Air (4th generation) A14 Bionic chip	S	128×128	600	9.37	1.67
		256×256	134	2.09	7.48
	L	128×128	412	8.70	1.80
		256×256	85	1.33	11.79
Apple iPhone 12 A14 Bionic chip	S	128×128	598	0.36	1.67
		256×256	144	2.26	6.92
	L	128×128	418	6.54	2.39
		256×256	92	1.43	10.89

Table 3. Benchmark of DyNCA WebGL implementations on different devices. This table shows the results for the Demo Type *Video Motion*. In this configuration multi-scale perception is enabled, and we set $T = 64$, i.e., we map 64 DyNCA steps to one video frame.

3. Training Details of DyNCA

We adopt the checkpoint pool trick introduced in [2]. We use a pool size of 256 and initialize the pool with zero-filled constant tensors as the seed. During training, we select a batch of states from the pool, evolve those states according to the DyNCA update rule, and then put the updated states back in the pool. Using a checkpoint pool allows the DyNCA model to see longer horizons of its PDE update rule, and helps with the long-term stability of the DyNCA evolution. Once every 8 epochs we replace one of the states in the pool with the initial seed state. This helps DyNCA to remember the seed state as the initial state of the PDE.

We also use the overflow loss introduced in [2, 3] to ensure training stability. Given a cell state of DyNCA $\mathbf{S}^t \in \mathbb{R}^{H\times W\times C}$, the overflow loss $\mathcal{L}_{over}$ of $\mathbf{S}$ is defined as:

$$\mathcal{L}_{over} = \frac{1}{HWC}\sum_{i}^{W}\sum_{j}^{H}\sum_{c}^{C}\left|\mathbf{S}^t_{i,j,c} - clip_{[-1,1]}(\mathbf{S}^t_{i,j,c})\right|$$

$$clip_{[a,b]}(x) = max(a, min(x, b)), \tag{1}$$

where $S_{i,j,c}$ is one of the entries of the DyNCA state tensor $\mathbf{S}$. Our final training objective is:

$$\mathcal{L}_{final} = \mathcal{L}_{\text{DyNCA}} + c\mathcal{L}_{over}, \tag{2}$$

where c is the overflow loss weight. Recall that $\mathcal{L}_{\text{DyNCA}}$, defined in page 4 section 4 of the main paper, consists of the DyNCA objectives for fitting the target appearance and target dynamic. The overflow loss is only computed on the last cell state in one training epoch.

In addition, we also adopt the gradient normalization trick introduced in [3], and normalize the L2 norm of each layer's gradient before applying them. Gradient normalization stabilizes the training and helps to prevent divergence during training. We train for 4000 epochs with an initial learning rate of 0.001, while multiplying the learning rate by 0.3 at 1000 and 2000 epochs.

For the optic flow prediction network, we re-implement the MSOENet introduced in [4] with PyTorch. We use the pre-trained TensorFlow model weights provided by [4] and transfer those weights to our PyTorch model.

In the following sections, we discuss the details of motion-learning training schemes from, first, motion from vector fields and, second, motion from videos.

3.1. Motion From Vector Field

For training DyNCA to synthesize motion according to a target vector field, we pick two synthesized images $\mathcal{I}^g_{t_1}, \mathcal{I}^g_{t_2}$ from the synthesized image sequence, and evaluate the loss $\mathcal{L}_{mvec}$ using these two images. Recall our loss functions from Equations (6, 7, 8) in page 5 of the main paper:

$$\mathcal{L}_{dir} = \frac{1}{HW}\sum_{i,j}\left(1 - \frac{U^g_{ij}\cdot U^t_{ij}}{\left\|U^g_{ij}\right\|_2\left\|U^t_{ij}\right\|_2}\right), \tag{3}$$

$$\mathcal{L}_{norm} = \frac{1}{HW}\sum_{i,j}\left|\frac{T}{t_2-t_1}\left\|U^g_{ij}\right\|_2 - \left\|U^t_{ij}\right\|_2\right|. \tag{4}$$

$$\mathcal{L}_{mvec} = (1.0 - min\{1.0, \mathcal{L}_{dir}\})\,\mathcal{L}_{norm} + \gamma\mathcal{L}_{dir}, \tag{5}$$

We use $T = 24$ in our motion from vector field experiments. We set t_1 to be the timestep of the selected states from the checkpoint pool and set $t_2 = t_1 + \mathcal{U}(32, 128)$, where $\mathcal{U}(32, 128)$ is a randomly chosen integer between 32 and 128. We use a seed size of 128×128 and a batch size of 4. We set the direction loss weight $\gamma = 1.5$, and the overflow loss weight $c = 100.0$. We set the initial motion loss weight to $\lambda = 10.0$ and then anneal this weight according to the appearance loss. We find this weight annealing helpful for fitting the motion from a target vector field.

In the next section, we provide the mathematical expressions of the target vector fields we use to train DyNCA.

3.1.1 Mathematical Expression of Target Vector Fields

We provide the mathematical definitions of each of the 12 hand-crafted motion vector fields we use to train the DyNCA models. Let $\tilde{U}^t_{i,j}$ be the un-normalized motion vector for each point (i, j) on the vector field image with size H, W, where $i \in [-\frac{W}{2}, \frac{W}{2}]$ and $j \in [-\frac{H}{2}, \frac{H}{2}]$. To obtain the final target vector field U^t, we L2-normalize $\tilde{U}^t$ using the following equation:

$$U^t = \frac{\tilde{U}^t}{\frac{1}{HW}\sum_{i,j}||\tilde{U}^t_{i,j}||_2} \tag{6}$$

- **Right.** $\tilde{U}^t_{i,j} = (cos(0°), sin(0°))$.
- **Up.** $\tilde{U}^t_{i,j} = (cos(270°), sin(270°))$.
- **Right acc. Right.** $\tilde{U}^t_{i,j} = (\frac{2i+W}{2} \times cos(0°), sin(0°))$.
- **Right acc. Down.** $\tilde{U}^t_{i,j} = (\frac{2j+H}{2} \times cos(0°), sin(0°))$.
- **Circular.** $\tilde{U}^t_{i,j} = (\frac{j}{\sqrt{H^2+W^2}}, \frac{-i}{\sqrt{H^2+W^2}})$.
- **Converge.** $\tilde{U}^t_{i,j} = (\frac{-i}{\sqrt{i^2+j^2}}, \frac{-j}{\sqrt{i^2+j^2}})$.
- **Diverge.** $\tilde{U}^t_{i,j} = (\frac{i}{\sqrt{i^2+j^2}}, \frac{j}{\sqrt{i^2+j^2}})$.
- **Hyperbolic.** $\tilde{U}^t_{i,j} = (\frac{j}{\sqrt{H^2+W^2}}, \frac{i}{\sqrt{H^2+W^2}})$.

- **2Block_X**.

$$\tilde{U}^t_{i,j} = \begin{cases} (cos(0°), sin(0°)), & j \geq 0, \\ (cos(180°), sin(180°)), & j < 0. \end{cases}$$

- **2Block_Y**.

$$\tilde{U}^t_{i,j} = \begin{cases} (cos(90°), sin(90°)), & j \geq 0, \\ (cos(270°), sin(270°)), & j < 0. \end{cases}$$

- **3Block**.

$$\tilde{U}^t_{i,j} = \begin{cases} (cos(0°), sin(0°)), & j \geq 0, \\ (cos(180°), sin(180°)), & i \geq 0, j < 0, \\ (cos(90°), sin(90°)), & i < 0, j < 0. \end{cases}$$

- **4Block**.

$$\tilde{U}^t_{i,j} = \begin{cases} (cos(0°), sin(0°)), & i \geq 0, j \geq 0 \\ (cos(270°), sin(270°)), & i \geq 0, j < 0 \\ (cos(90°), sin(90°)), & i < 0, j \geq 0, \\ (cos(180°), sin(180°)), & i < 0, j < 0. \end{cases}$$

3.2. Motion From Video

3.2.1 Dynamic Texture Synthesis

We use texture videos from the dataset introduced in [4], employing all 12 frames in each video for training. When learning the motion from target videos, DyNCA iterates between 80 and 144 steps at each training epoch. The overflow loss weight c is 1.0. When training with a seed size of 128×128, we set the batch size to 4. For training with 256×256 seed size, we set the batch size to 3 due to GPU memory limits.

DyNCA is sensitive to λ, the weight of the motion loss. Small λ's can cause wrong motion or fixed frames while large values can lead to failed texture fitting. To automatically set the weight for motion loss, we first train DyNCA for 1000 epochs with an empirical motion loss weight of 5.0. Then we record the median of the motion loss, denoted as $\mathcal{L}^{median}_{mvid}$, and re-initialize DyNCA and the pool. We find a roughly linear relationship between the median loss and the proper weight and set the weight according to it. We give the concrete values of λ for each DyNCA configuration as follows:

- S-128, L-128: $5.82 \times \mathcal{L}^{median}_{mvid} - 1.05$.
- S-256, L-256: $6.04 \times \mathcal{L}^{median}_{mvid} - 2.17$.

Finally, we train DyNCA with extra 4000 epochs.

3.2.2 Choice of NCA Interval in Video Motion

When DyNCA learns target motions from videos, it uses T steps to fit one frame of the video. Larger T can contribute to a better fitting of the target motion and appearance since DyNCA acts as a discrete-time PDE and thus more time steps can make the result more precise. We show results with DyNCA intervals of 32 compared to 64, which is what we use in the experiments, in Figure 4.

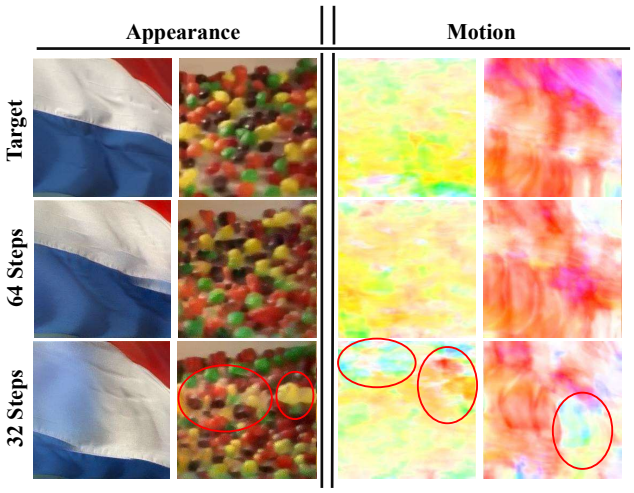

Figure 4. Comparison between training with 32 and 64 DyNCA time intervals for fitting one frame. Fewer DyNCA steps lowers the quality of the synthesized frame and leads to wrong motion. The two columns of flow images come from water_3 and flag_2 in the dataset of [4]

We can see that with fewer DyNCA steps, the quality of the texture synthesis decreases and artifacts occur. We also observe that DyNCA might generate wrong motions than the target video after training with a small number of intervals. Therefore, we select the maximum DyNCA steps in DyNCA-L-256^2 to train on a single Nvidia-A100 without exceeding memory, which is limited to 64. We use the same setting across all DyNCA configurations to ensure consistency.

3.2.3 Dynamic Style Transfer

We use the videos in the dataset introduced in [4] as the sources of target motions and use the images in figure 5 as the sources of target appearances. We use the same training settings for dynamic style transfer as in dynamic texture synthesis (DyTS), except for the weight setting scheme. The target appearance and target motion in dynamic style transfer might be incompatible with each other. Hence, the automatic weight setting scheme in DyTS can generate incorrect weight values, leading to low-quality results. Therefore, we manually set the motion loss weight λ in dynamic

Target Dynamic	Target Appearance	Weight
flames	cartoon_fire_1	2.0
	cartoon_fire_2	3.0
	cartoon_fire_4	2.0
	cartoon_fire_5	0.5
	cartoon_fire_6	2.0
fireplace_1	cartoon_fire_2	5.0
	cartoon_fire_4	8.0
	cartoon_fire_6	8.0
fireplace_2	cartoon_fire_2	10.0
	cartoon_fire_3	9.0
	cartoon_fire_4	5.0
	cartoon_fire_6	9.0
sea_2	cartoon_water_1	3.0
	cartoon_water_2	3.0
	cartoon_water_3	6.0
	cartoon_water_4	3.0
water_3	cartoon_water_1	3.0
	cartoon_water_3	0.5
	cartoon_water_4	0.5

Table 4. Weight settings of dynamic style transfer with DyNCA-L-256. All target dynamics come from the dataset introduced in [4]. Target appearance images refer to figure 5.

style transfer experiments. Concrete values are given in table 4.

3.3. Limitations

Although we have a scheme to automatically set the motion loss weight λ, we observe that the automatic weight leads to poor results in 7 of the 59 texture videos in the dataset introduced in [4]. This can cause low-quality synthesized videos or diverging synthesis. Diverging synthesis means that DyNCA cannot generate meaningful patterns during video synthesis after a certain number of timesteps.

To solve this problem, we manually set the motion loss weight λ for those 7 videos on both DyNCA-S and DyNCA-L configurations. We show the concrete values of the manually set weights for each of these 7 dynamic texture videos in Table 5, and Table 6. Table 5 shows both the manual and automatically set weights for 2 of the 7 videos that yield low-quality results when we use the automatic weight setting scheme. Table 6 shows both the manual and automatically set weights for 5 of the 7 videos that yield diverging results when we use the automatic weight setting scheme.

NCA Configs	Weight Scheme	flames	sea_2
S-256	Automatic	5.5	2.1
	Manual	3.0	4.0
L-256	Automatic	5.7	2.0
	Manual	2.0	4.0

Table 5. Two low-quality textures when using automatic weight settings. After manually tuning the weights, DyNCA can synthesize realistic texture videos.

4. Quantitative Evaluation of Multi-scale Perception Ablation Study

In the main paper, we show qualitative results of the ablation study for multi-scale perception. We further quantitatively evaluate the effect of multi-scale perception by observing $\mathcal{L}_{appr}$ and $\mathcal{L}_{mvid}$ during video synthesis after training. In video synthesis, DyNCA generates one frame every T steps, where $T = 64$. Denote two consecutive frames as $\mathcal{I}^g_t, \mathcal{I}^g_{t+T}$, where t is the current time step. We compute $\mathcal{L}_{appr}$ for $\mathcal{I}^g_t$ and all frames in the target video. Then we average the result to obtain $\mathcal{L}^t_{appr}$. The final test $\mathcal{L}_{appr}$ is obtained via averaging over all time steps. We also adopt this scheme for computing $\mathcal{L}_{mvid}$, where the inputs are $\mathcal{I}^g_t, \mathcal{I}^g_{t+T}$.

In Table 7, we can see that multi-scale perception helps decrease the loss values overall. When DyNCA has more parameters and the resolution of seeds is large (DyNCA-L-256), the effect becomes more obvious.

5. Ablation Study for Positional Encoding

We demonstrate qualitatively and quantitatively that our proposed positional encoding improves the performance of DyNCA.

5.1. Motion from Vector Field

In the main paper, we quantitatively show that using Replicate Padding + Cartesian Positional Encoding (CPE) achieves better results compared to other padding strategies without positional encoding. The results provided in the paper were averaged over 10 different target appearances and 4 different structured motion vector fields (Circular, Converge, Diverge, and Hyperbolic). We provide both qualitative and quantitative results for other motion vector fields. We compare DyNCA with CPE to 3 different baselines that use different padding strategies (Zero-Padding, Replicate-Padding, and Circular-Padding) but do not utilize positional encoding. We use the DyNCA-S configuration and train the models with a seed size of 128×128.

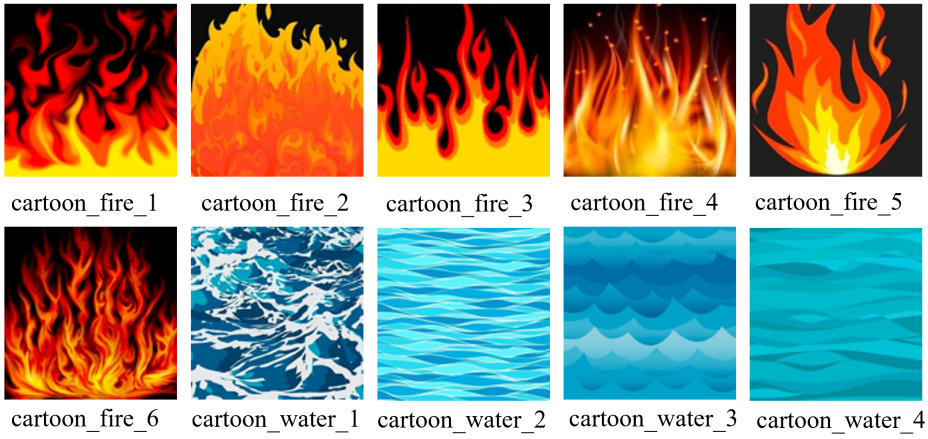

Figure 5. Target appearance images used in dynamic style transfer with DyNCA. Our dynamic style transfer synthesized videos are available at https://dynca.github.io/#style_transfer.

NCA Configs	Weight Scheme	ants	sky_clouds_1	smoke_2	smoke_3	calm_water_2
S	Automatic	4.6	10.0	5.8	7.9	6.7
	Manual	0.2	0.25	0.1	0.5	1.0
L	Automatic	4.1	10.0	6.8	6.5	6.6
	Manual	0.2	0.25	0.1	1.0	1.0

Table 6. Five textures that will diverge during video synthesis when using automatic weight settings. After manually tuning the weight, DyNCA can robustly synthesize realistic texture videos.

NCA Configs	$\mathcal{L}_{appr}$		$\mathcal{L}_{mvid}$	
	Multi	Single	Multi	Single
DyNCA-S-128	3.1816	**3.1762**	0.1707	**0.1670**
DyNCA-L-128	**3.1267**	3.1789	**0.1641**	0.1654
DyNCA-S-256	**2.5536**	2.6343	**0.1959**	0.2008
DyNCA-L-256	**2.5633**	2.6918	**0.1901**	0.1979

Table 7. Loss values during video synthesis after training. Multi-scale perception helps decrease the values of $\mathcal{L}_{appr}$ and $\mathcal{L}_{mvid}$, thus contributing to DyNCA better fitting the target appearance and motion.

5.1.1 Qualitative Analysis

Figure 6 shows the qualitative results of our ablation study for positional encoding. The first two rows in Figure 6 illustrate the 12 different target vector fields we used in our training. We define a **Structured Vector Field** as a vector field in which either the direction or the magnitude of the motion is position-dependent. For example, in the *Right acc. Right* and *Right acc. Down* vector fields, the magnitude of the motion depends on the position. In the *Converge*, *Diverge*, *2Block_X*, *2Block_Y*, *3Block*, and *4Block* vector fields, the direction of the motion is position-dependent. And finally, in the *Circular*, and *Hyperbolic* vector fields, both the motion magnitude and motion direction depend on the position.

Figure 6 shows snapshots of the optic flow for the videos synthesized by the baselines and the final DyNCA configuration utilizing CPE. We can observe that for simple vector fields, such as *Right* and *Up*, all of the baselines fit the motion and generate acceptable results. However, when the target vector field becomes more structured and complex, the baselines fail to learn the correct motion. For example, all the baselines fail when the motion magnitude is position-dependent. Although the DyNCA baselines with zero-padding and replicate padding are able to fit some structured target vector fields such as *Diverge* and *2Block_X*, we can observe that after resizing the input seed to 256×256 the baseline fails to generate the correct motion.

5.1.2 Quantitative Analysis

For the quantitative results, we use the same DyNCA-S configuration and train the baselines for all 12 different target fields on the following 10 target appearance textures: *bubbly_0101*, *chequered_0121*, *fi-*

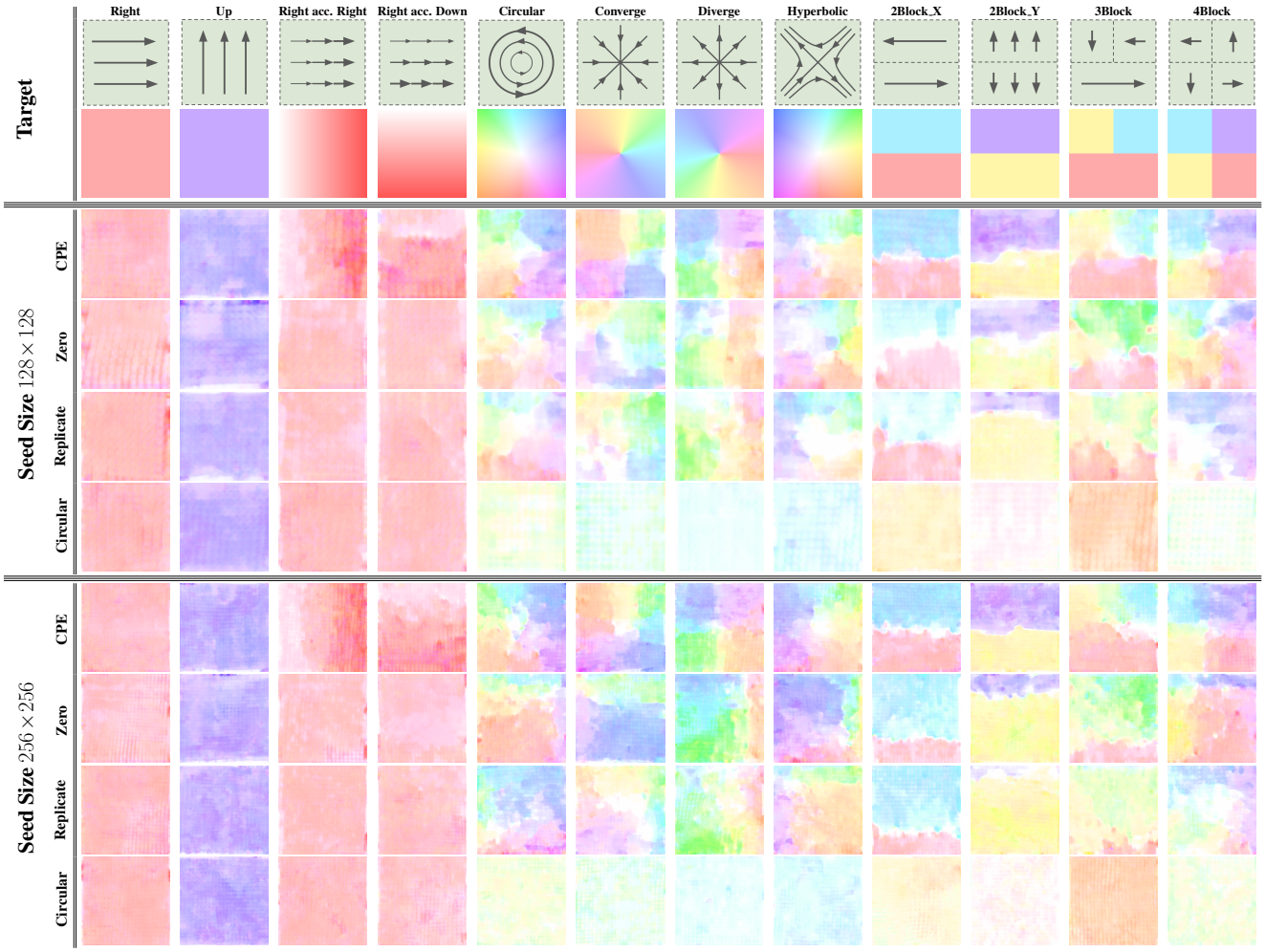

Figure 6. Qualitative comparison of DyNCA baselines with different padding strategies and with the proposed DyNCA model with Cartesian Positional Encoding (CPE). We observe that the DyNCA with CPE better fits the target motion. Moreover, the model using CPE is able to synthesize the correct motion regardless of the seed size. The target appearance used is chequered_0121 from the DTD [1] dataset.

brous_0145, *cracked_0085*, *interlaced_0172*, *water_3*, *smoke_2*, *smoke_plume_1*, *calm_water_4*, and *sea_2*. To evaluate the motion direction loss $\mathcal{L}_{dir}$ and motion magnitude loss $\mathcal{L}_{norm}$, we synthesize a 330 frame video and exclude the first 30 frames. We then feed the remaining 300 frames into the optic flow prediction network provided in [4] and compare the estimated optic flow with the target vector fields. We average the losses over the 300 frames and over all of the target appearances and report the results in Table 8. The results show that for simpler vector fields such as *Right* and *Up*, circular padding works better than CPE. However, for more complex and structured vector fields, CPE achieves better results both in terms of motion direction loss $\mathcal{L}_{dir}$ and motion magnitude loss $\mathcal{L}_{norm}$.

5.2. Motion from Video

5.2.1 Qualitative Analysis

Without positional encoding, we observe several failed or distorted synthesized video frames, as shown in Figure 7. Training without CPE can cause artifacts on synthesized frames and can lead to texture-less images of low quality.

5.2.2 Quantitative Analysis

In Table 9, we record the loss values during video synthesis obtained from DyNCA trained with and without Cartesian Positional Encoding(CPE). The method for obtaining $\mathcal{L}_{appr}$ and $\mathcal{L}_{mvid}$ is the same as in the quantitative evaluation of the multi-scale perception ablation study, namely average across all ground-truth frames.

Seed Size	Loss	Config	Right	Up	Right acc. Right	Right acc. Down	Circular	Converge	Diverge	Hyperbolic	2Block_X	2Block_Y	3Block	4Block
128 × 128	$\mathcal{L}_{dir}$	CPE	0.027	0.032	0.055	0.049	**0.048**	**0.072**	**0.075**	**0.053**	**0.102**	**0.085**	**0.107**	**0.104**
		Zero	0.035	0.032	0.052	0.059	0.124	0.241	0.165	0.116	0.170	0.169	0.326	0.261
		Replicate	0.024	0.033	0.054	0.047	0.241	0.339	0.344	0.345	0.289	0.365	0.336	0.452
		Circular	**0.022**	**0.027**	**0.025**	**0.025**	0.979	0.997	0.994	0.985	0.985	0.998	0.659	0.988
	$\mathcal{L}_{norm}$	CPE	0.227	0.183	**0.251**	**0.246**	**0.233**	**0.230**	**0.229**	**0.249**	**0.268**	**0.232**	0.251	**0.229**
		Zero	0.238	0.182	0.339	0.303	0.300	0.312	0.263	0.307	0.318	0.277	0.279	0.273
		Replicate	**0.201**	0.190	0.378	0.330	0.371	0.301	0.344	0.438	0.341	0.350	0.284	0.330
		Circular	0.213	**0.153**	0.551	0.555	0.618	0.651	0.531	0.759	0.837	0.819	**0.179**	0.651
256 × 256	$\mathcal{L}_{dir}$	CPE	0.021	0.026	0.044	0.045	**0.050**	**0.059**	**0.058**	**0.047**	**0.089**	**0.068**	**0.095**	**0.098**
		Zero	0.023	0.024	0.037	0.048	0.384	0.491	0.446	0.322	0.543	0.474	0.441	0.503
		Replicate	0.020	0.023	0.058	0.040	0.467	0.494	0.457	0.492	0.579	0.617	0.483	0.555
		Circular	**0.018**	**0.021**	**0.021**	**0.020**	0.981	0.998	0.994	0.999	0.995	1.010	0.662	0.994
	$\mathcal{L}_{norm}$	CPE	0.202	0.167	**0.215**	**0.210**	**0.221**	**0.210**	**0.217**	**0.222**	**0.236**	**0.208**	0.231	**0.220**
		Zero	0.214	0.169	0.411	0.372	0.423	0.283	0.254	0.364	0.254	0.228	0.255	0.340
		Replicate	**0.185**	0.169	0.410	0.444	0.408	0.317	0.329	0.533	0.279	0.295	0.287	0.329
		Circular	0.196	**0.139**	0.555	0.540	0.634	0.634	0.518	0.926	0.839	0.915	**0.181**	0.798

Table 8. Quantitative comparison of DyNCA baselines with different padding strategies with the proposed DyNCA model with Cartesian Positional Encoding (CPE). We observe that DyNCA with CPE can better fit the target motion.

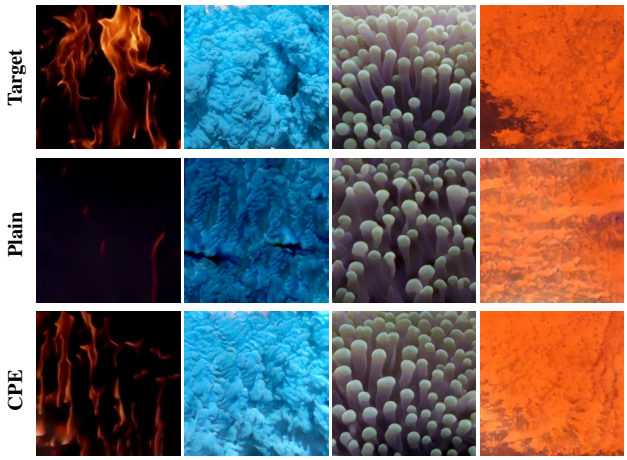

Figure 7. Comparison between training with and without Cartesian Positional Encoding (CPE). The first row shows the target appearance texture. The results without CPE (second row) have less texture or artifacts compared with the third row (with CPE). The results are obtained with DyNCA-S-256.

In all DyNCA settings, positional encoding helps decrease the test loss during video synthesis, indicating the importance of our proposed positional encoding method.

6. More Results on Dynamic Texture Synthesis

We show more qualitative results on dynamic texture synthesis in Figures 8, 9, and 10.

NCA Configs	$\mathcal{L}_{appr}$		$\mathcal{L}_{mvid}$	
	CPE	Plain	CPE	Plain
DyNCA-S-128	**3.1762**	4.3809	**0.1670**	0.1874
DyNCA-L-128	**3.1789**	3.9818	**0.1654**	0.1834
DyNCA-S-256	**2.5536**	2.9358	**0.1959**	0.2102
DyNCA-L-256	**2.5633**	2.8756	**0.1901**	0.1980

Table 9. Loss values during video synthesis after training. "Plain" refers to not adding positional encoding to cell states. CPE contributes to a better fitting of target appearance and motion.

7. More Results on Dynamic Style Transfer

We show more results on dynamic style transfer in Figures 11 and 12. The synthesized videos are available at our website https://dynca.github.io/#style_transfer.

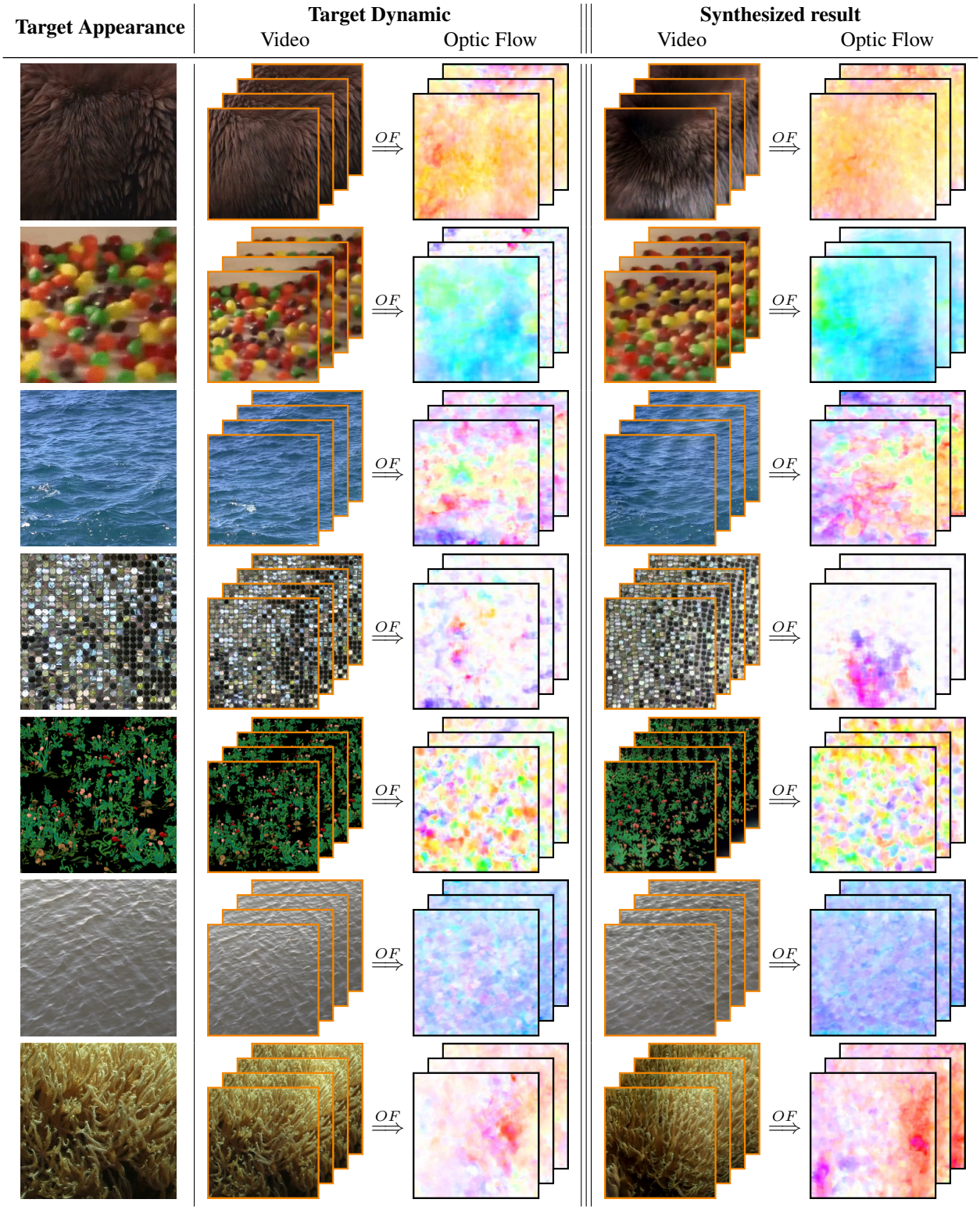

Figure 8. Results of dynamic texture synthesis with DyNCA-L-256.

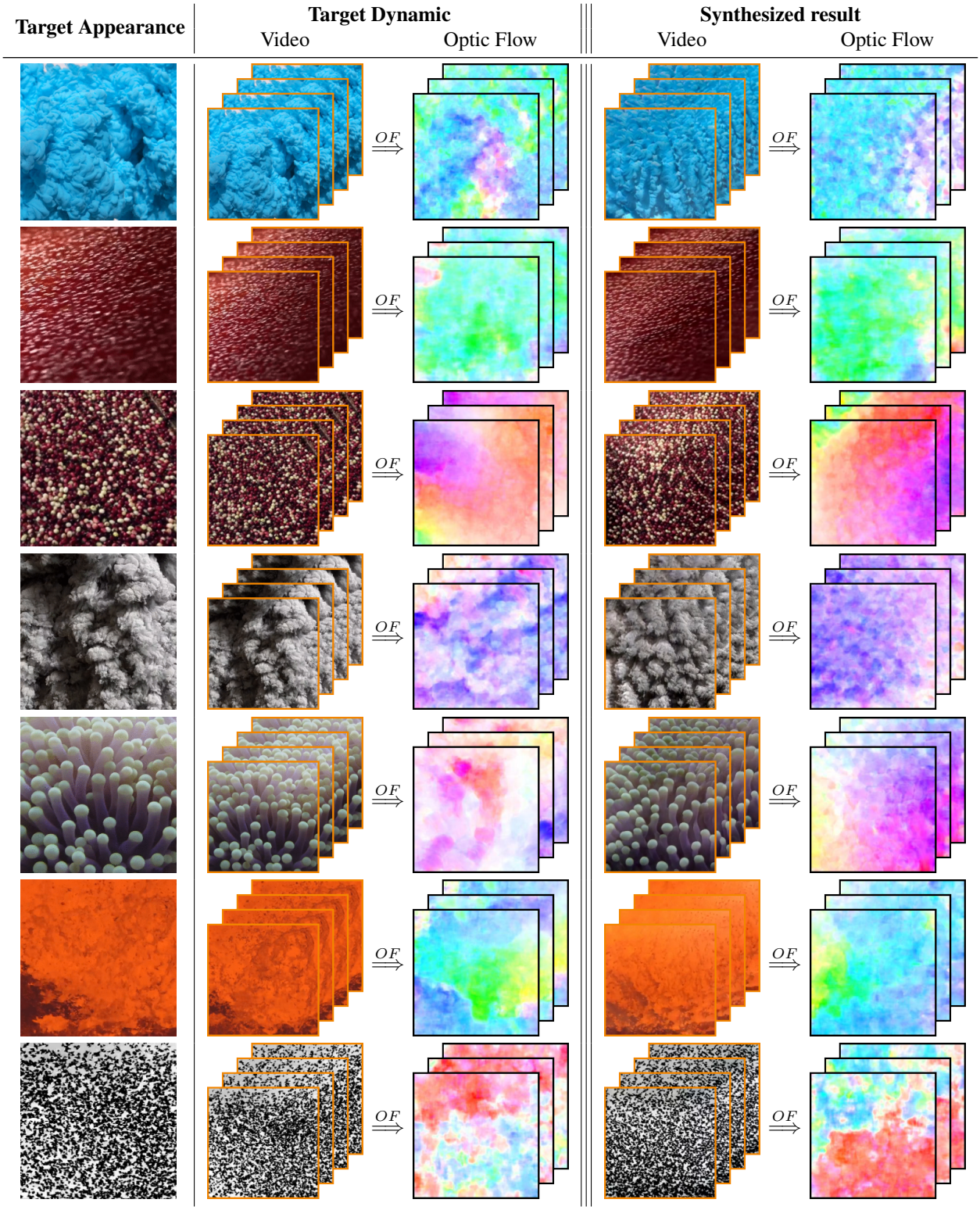

Figure 9. Results of dynamic texture synthesis with DyNCA-L-256.

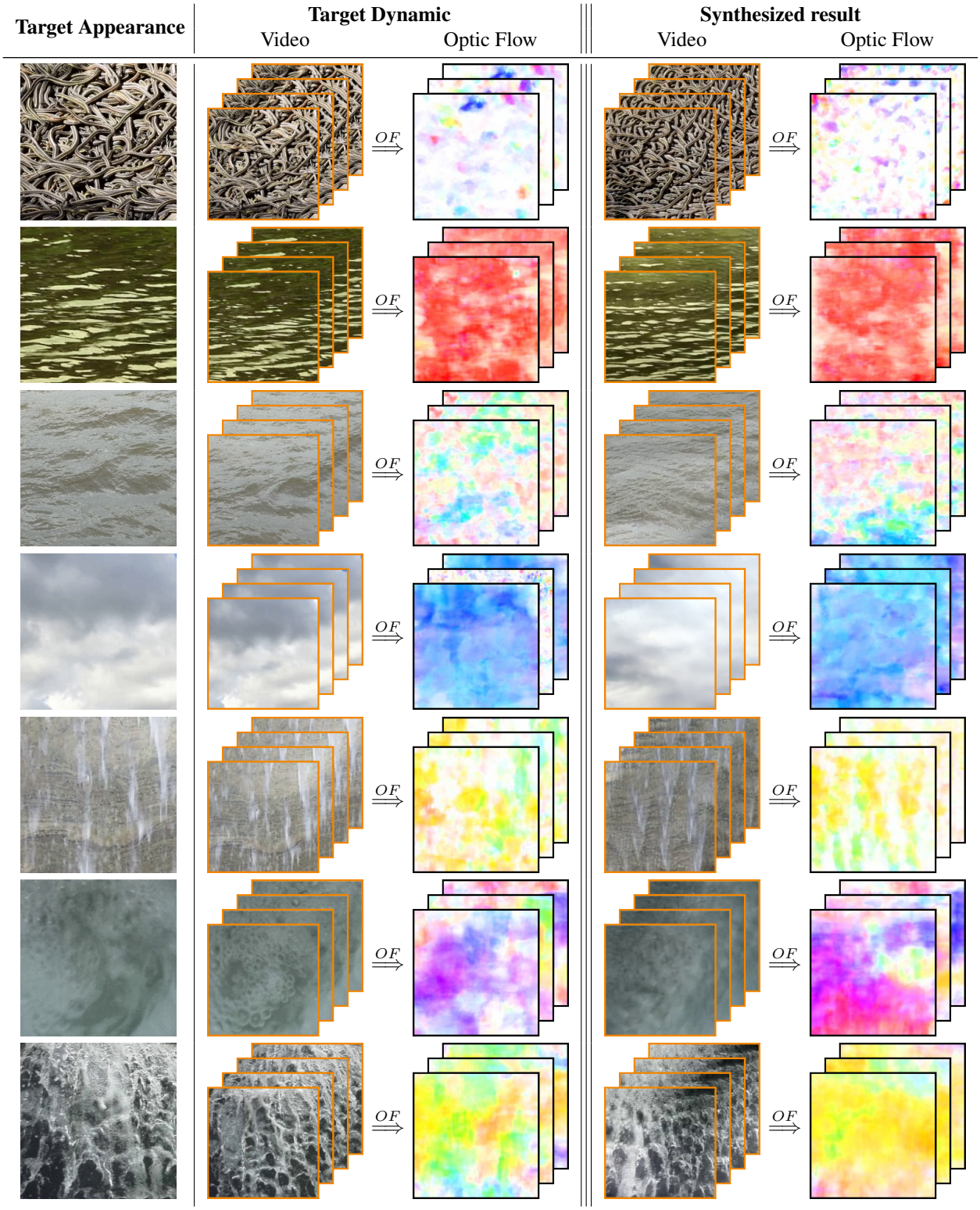

Figure 10. Results of dynamic texture synthesis with DyNCA-L-256.

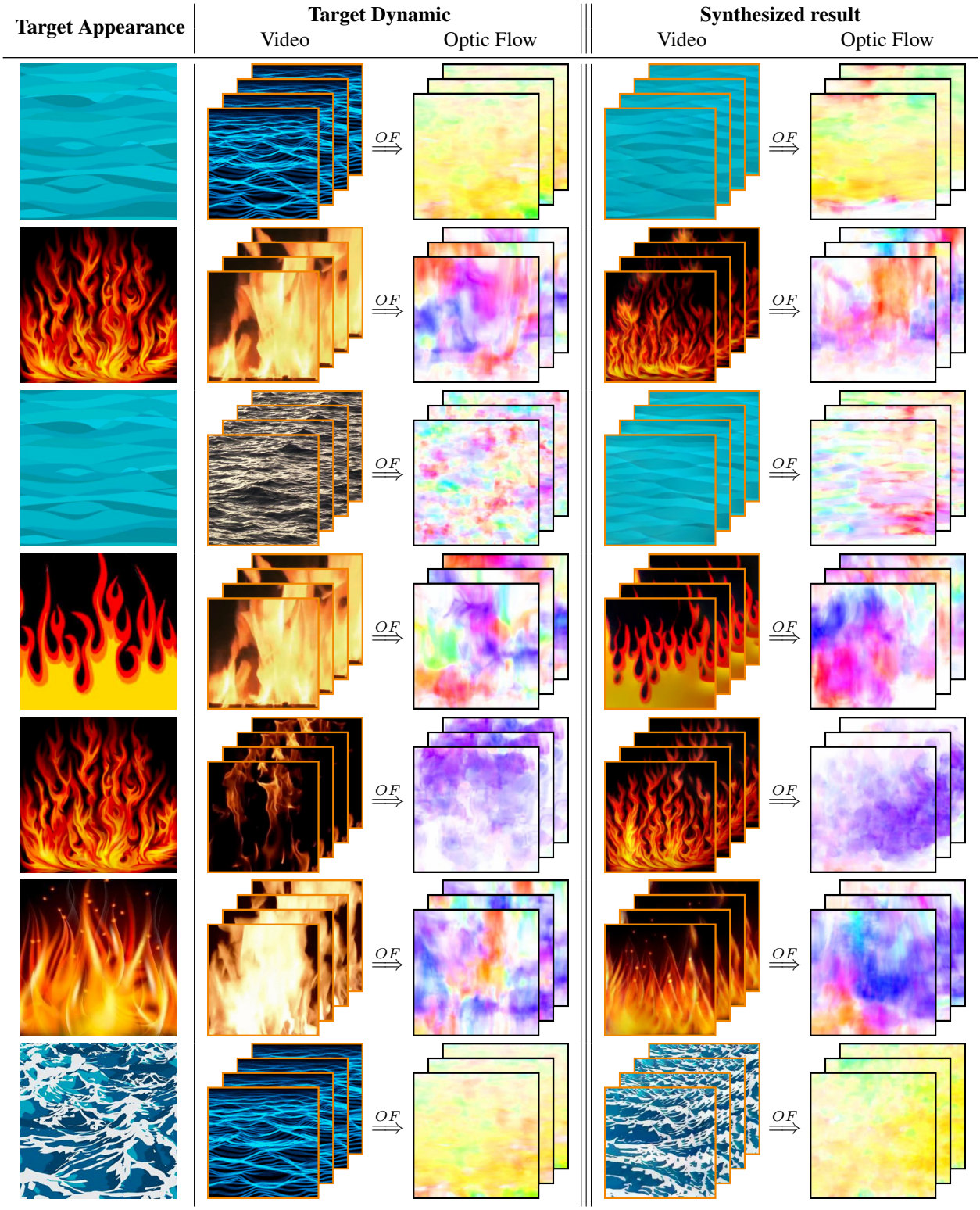

Figure 11. Results of dynamic style transfer with DyNCA-L-256.

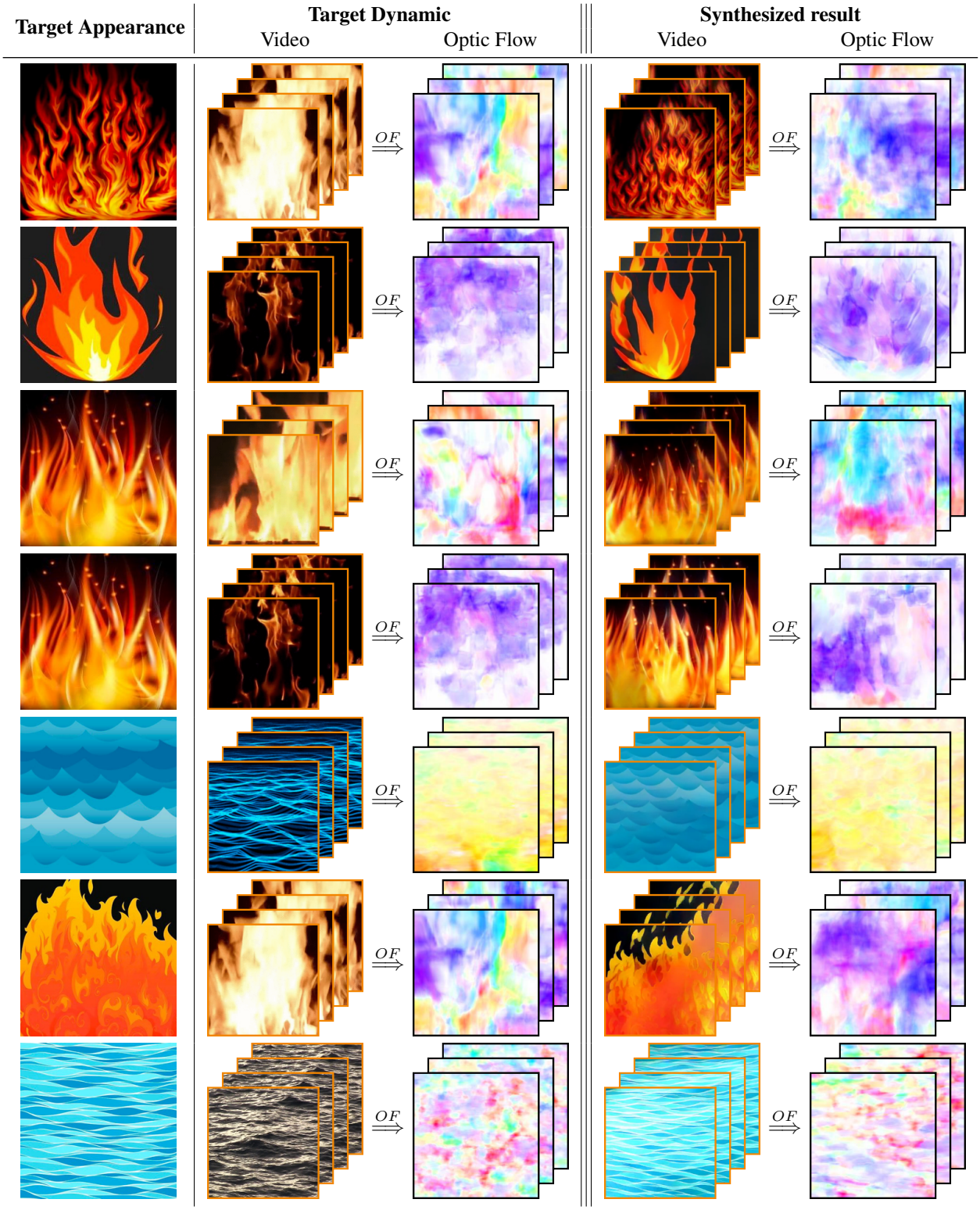

Figure 12. Results of dynamic style transfer with DyNCA-L-256.

References

[1] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014. 11

[2] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 2020. https://distill.pub/2020/growing-ca. 7

[3] Eyvind Niklasson, Alexander Mordvintsev, Ettore Randazzo, and Michael Levin. Self-organising textures. *Distill*, 6(2):e00027–003, 2021. 2, 7

[4] Matthew Tesfaldet, Marcus A Brubaker, and Konstantinos G Derpanis. Two-stream convolutional networks for dynamic texture synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6703–6712, 2018. 1, 2, 7, 8, 9, 11

[5] Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu. Synthesizing dynamic patterns by spatial-temporal generative convnet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7093–7101, 2017. 1, 2