# Playable Environments: Video Manipulation in Space and Time
# Supplementary Material

Willi Menapace[*]
University of Trento

Stéphane Lathuilière[†]
LTCI, Télécom Paris
Institut Polytechnique de Paris

Aliaksandr Siarohin
University of Trento

Christian Theobalt[†]
MPI for Informatics, SIC

Sergey Tulyakov[†]
Snap Inc.

Vladislav Golyanik[†]
MPI for Informatics, SIC

Elisa Ricci[†]
University of Trento
Fondazione Bruno Kessler

In this Supplementary Material, we discuss additional implementation details (see Sec. 1), training details (see Sec. 2), provide details on the proposed datasets (see Sec. 3) and show additional experimental results (see Sec. 4). This Supplementary Material is complemented by a website (see willi-menapace.github.io/playable-environments-website) showing additional video qualitative results such as sequences produced interactively by users, camera manipulation and style manipulation results. We also submit our code for review which will be made publicly available.

## 1. Implementation Details

In this section, we discuss additional implementation details for our method. Sec. 1.1 introduces NeRF [10], Sec. 1.2 discusses the architecture of the models adopted in our method, Sec. 1.3 discusses the implementation details of our feature renderer ConvNet, Sec. 1.4 describes the organization of the objects of which the environment is composed, Sec. 1.6 describes the sharing strategies that are used to model different instances of the same object class and Sec. 1.7 describes details regarding batch normalization and pooling operations in the action module.

### 1.1. Introduction to NeRF

NeRF [10] represents scenes as radiance fields: a 5D continuous function that maps a 3D position and a 2D viewing direction to an emitted color $c$ and an opacity $\sigma$. NeRF parametrizes such function as an MLP.

Given a desired camera perspective, the associated image can be rendered exploiting the radiance field representation [10]. A ray $r$ is traced through each pixel and the associated color is obtained by integration:

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t))dt, \tag{1}$$

with

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(r(s))ds\right), \tag{2}$$

where $T(t)$ denotes the accumulated transmittance, and $t_n$ and $t_f$ represent the minimum and maximum distance from the camera in which the integration is performed. In practice, the integral can be approximated through quadrature [10]. $N$ positions along each ray $r$ are sampled and the associated color is computed as

$$\hat{C}(r) = \sum_{i=1}^{N} T_i(1 - e^{-\sigma_i \delta_i})c_i, \tag{3}$$

with

$$T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j), \tag{4}$$

where $c$ and $\sigma$ are obtained by querying the MLP and $\delta$ denotes the distance between successive samples.

In order to ease the learning of high frequency details, NeRF employs positional encodings of the 3D position and viewing direction as input to the underlying MLP. In addition, *Mildenhall et al.* [10] propose a stratified sampling approach which exploits a *coarse* neural radiance field to locate portions of the ray corresponding to visible surfaces and allocates additional samples to the neighborhood of such regions. A *fine* neural radiance field is used to compute the final image using both the initial and the additional sampled locations.

[*]This work was partially done while interning at MPI for Informatics
[†]Equal senior contribution

The model is trained using L2 distance between the reconstructed and the ground-truth color value for each sampled ray.

## 1.2. Architecture Details

In this section, we discuss the details of the models employed in our framework. Fig. 1 shows the models employed on the *Tennis* dataset.

We make use of positional encodings [10] for both $V$ and the bending network $B$. We use 10 and 6 octaves, respectively, and use the concatenation of the original vector and the encodings as input. Following [12], we gradually introduce the positional encodings in $B$ during the first 60.000 training iterations. No viewing direction is given as input to $B$ and $V$ to avoid artifacts when the environment is rendered from a camera pose outside of the training distribution. In addition, to reduce memory consumption, we do not make use of the hierarchical sampling scheme (see Sec. 1.1).

We model the action network $A$ following [9] and make use of gumbel softmax sampling [6] to obtain discrete action representations. In addition, to foster $A$ in predicting actions that are associated to movement, we use only object positions $x_t$ as input rather than the complete environment state $s_t$.

The temporal discriminator $D$ receives as input the object positions $x$ and object poses $\pi$ and is further conditioned on the actions $a$ and action variabilities $v$ inferred on the ground-truth sequence.

We modulate the number of residual blocks in $E$ based on the expected size of the input image crop.

## 1.3. Feature Renderer Details

We propose to render feature maps at multiple resolutions into the final image using a ConvNet feature renderer $F$. Our ConvNet accepts feature maps $\{f_i\}_{i=1}^{l}$ at $l$ different resolutions, each associated to a downsampling factor $d_i$. We obtain feature map $f_i$ by integration of the features sampled along the rays corresponding to pixel $I_{mn}$ in the original image, such that

$$m \in \left\{ \frac{d_i}{2} + 1 + kd_i \right\}_{k=0}^{\frac{h}{d_i}-1}, n \in \left\{ \frac{d_i}{2} + 1 + kd_i \right\}_{k=0}^{\frac{w}{d_i}-1}, \tag{5}$$

where $h$ and $w$ are the image height and width and indexes are expressed starting from 1. The process is illustrated in Fig. 2. Note that each ray is sampled on a grid where the points are at a vertical and horizontal distance of $d_i$ from one another and at distance $d_i/2$ from the border (see left of Fig. 2 for a visualization of the sampled positions). Note also that the ray is sampled at the pixel location in the original image that corresponds to the center of the associated local patch that will be synthesized by $F$.

In our implementation, we render two separate feature maps at downsampling factors 8x and 4x (see Fig. 1) to capture details at different scales. Note that this approach allows rendering images using 12.8x fewer rays than NeRF approaches that render each pixel with a separate ray, allowing for important reductions in the use of memory and in computational complexity and enables the model to render large image patches at training time. We experiment with greater downsampling factors, but find that further increasing them causes 3D consistency artifacts.

In the presence of calibration and localization noise during training, we observe that NeRF models tend to generate blurry results which are particularly evident on dynamic objects where calibration and localization errors are compounded with errors in the estimation of the object pose. We ascribe this phenomenon to the use of reconstruction losses based on L2 distance which, in the presence of a color mismatch between the ground truth and the reconstructed pixel caused by input noise, favors the prediction of intermediate color values, generating blur. Using a feature renderer provides two advantages over the traditional approach. First, thanks to convolutional filters, the prediction for the feature associated to the current position can take into account the values of neighboring positions, enabling the model to produce a coherent output. Second, the possibility to render large patches makes it possible to adopt losses different from L2 distance, such as the perceptual loss of *Johnson et al.* [7] which penalizes more effectively implausible predictions such as the ones containing blur.

## 1.4. Object Configurations

On the *Minecraft* dataset, we model the scene using four objects. The first object models the static scene elements close to the players, the second object models the distant objects and the other two objects model the players. For each ray, we sample 16 positions for the first model, 1 position for the second model and 32 positions for each of the players.

On the *Tennis* dataset, we note that camera movement in the input dataset is mostly limited to camera rotations. Recovering depth information for the static objects is thus an ill-posed problem which requires prior knowledge. Note that this problem is not present for the players since their change of position with respect to the camera allows the learning of depth information. We thus adopt the following objects to model the environment: an object to model the tennis field which is bounded by a box $\beta$ that does not rise above ground level, an object to model the backplate of the

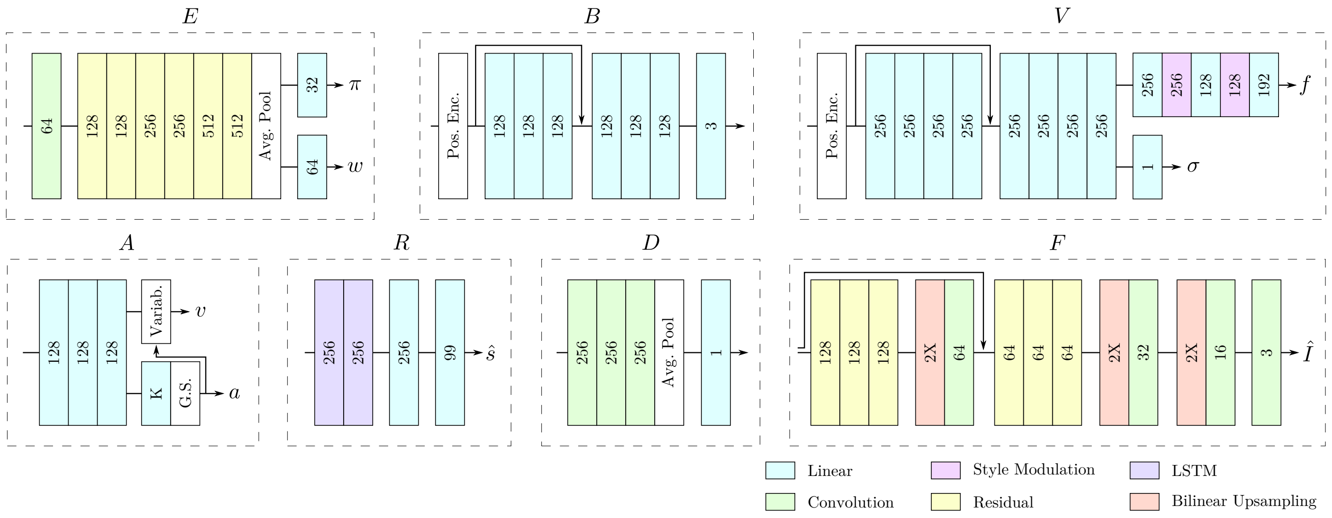


Figure 1. Architecture of the models employed by our method on the *Tennis* dataset. The number in each block indicates the number of output features or the upsampling factor in the case of upsampling operations. Merging of arrows indicates concatenation. The architecture for the encoder $E$ refers to the encoder for the static object models. We adopt a smaller encoder for the players. *G.S.* indicates Gumbel Softmax [6]; *Pos. Enc.* indicates positional encoding.

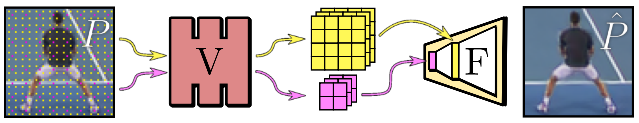


Figure 2. Representation of the feature rendering process. We sample points on rays arranged in a grid corresponding to feature map locations, then the sampled points are rendered into feature maps using our NeRF pipeline. The process is repeated to produce feature maps at different resolutions. The feature rendered $F$ uses the rendered feature maps to reconstruct the original image.

tennis field, which is bounded by its box $\beta$ to be a planar surface, and two objects for the players. For each ray, we sample 4 positions for the objects modeling the static scene and 32 positions for each player.

## 1.5. Minecraft Background Modeling

The *Minecraft* dataset features a challenging environment with distant visible objects. NeRF++ [14] employs an inverted sphere scene background representation for modeling distant objects. While effective, this parametrization requires a large number of samples to model distant objects, which increases memory consumption. To address this issue, we propose to model background objects using a spheric background representation which requires a single sample for each ray. In particular, we model the spheric background with as an MLP $f = V(d, o)$, receiving as input the direction $d$ and the origin $o$ of the ray, and returning as output the associated feature. We consider the opacity $\sigma$ of the associated feature to always be 1. The parametrization on the ray origin allows the model to simulate the effects of depth on the background objects without requiring multiple samples.

## 1.6. Model Sharing

Instead of using a synthesis and an action module specific to each object, we note that objects corresponding to the same class can, in principle, be represented by the same model. In particular, in the *Minecraft* dataset, the two players have completely symmetric characteristics, so we model both using a single shared model. While the same observation can be made for the case of the players in the *Tennis* dataset, we found that the synthesis module fails if the same model is used for both players. We explain this behavior by observing that on the *Tennis* dataset the player closest to the camera is always observed from the back, while the player further from the camera is always observed from the front. This asymmetry makes it harder for a model to learn a unified player representation, so we model the players with separate models.

## 1.7. Masked Batch Normalization and Pooling

In a video sequence, detection may fail in some frames. In the action module, the environment state corresponding to frames with missing detections and the successive ones are replaced with placeholder values. As discussed in Sec. 2.2, loss masking is used to prevent effects of placeholder values on training. Nevertheless, the presence of placeholder values may still alter the behavior of the action module by affecting the estimation of batch statistics in batch normalization layers and by affecting the behavior of operations such as global average pooling. To prevent potential undesired effects on training, in the action module we adopt masked versions of batch normalization and of pooling operations that operate by computing statistics only on non-placeholder values.

## 2. Training details

In this section, we describe training details for the synthesis (see Sec. 2.1) and the action modules (see Sec. 2.2).

### 2.1. Synthesis Module Training Details

We optimize the synthesis module parameters using Adam [8] as optimizer and learning rate $5e-4$. The model is trained for 300.000 iterations and the learning rate is exponentially decayed until $5e-5$ at the end of training. We employ a batch size of 8 video sequences, each with 3 or 4 frames for *Minecraft* and *Tennis*, respectively. To better disentangle style from pose (see Sec. 3.3 on the main paper) we do not use consecutive frames for each sequence, but skip 4 and 100 frames between each selected frame, respectively, for *Minecraft* and *Tennis*.

In all the experiments, we consider input images of size 512x288 and we render patches of size 192x192 and 256x256, respectively, for *Minecraft* and *Tennis* during training. To improve the quality of playable objects, we sample with increased frequency patches that contain the players.

The total loss is the weighted sum of the perceptual and L2 reconstruction losses in pixel space. We assign a weight of 1.0 to the L2 reconstruction loss and a weight of 0.1 to the perceptual loss term.

The synthesis module is trained on 4x Nvidia RTX 8000. We also train models on 1x Nvidia RTX 8000 on the *Tennis* dataset using a reduced rendered patch size of 160x160 pixels with a small reduction in image quality.

**Feature renderer pretraining.** While it is possible to learn the feature renderer network $F$ from scratch, we find it beneficial to start training from pretrained weights since this guides the composable NeRF model towards learning features encodings whose mapping to the image space is already known. We obtain these weights with a pretraining process. We add a temporary encoder ConvNet to $F$ and train it as an autoencoder, using the same combination of perceptual and L2 losses used for training of the synthesis module. To avoid disruption of the learned features in the early stage of training of the synthesis module, we freeze $F$ during the first training iterations of the full model and unfreeze it once the composable NeRF model features become close to the ones the temporary ConvNet encoder would have produced.

### 2.2. Action Module Training Details

For the action module, we make use of Adam [8] as optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$, and use a learning rate of $5e-4$. We train the model for 300.000 iterations with an exponentially decayed learning rate which reaches the value of $5e-5$ at the end of training. The action module and the discriminator are optimized in alternation. We regularize the discriminator training using spectral normalization [11].

We note that at inference time our autoregressive dynamics network receives as input sequences of reconstructed environment states $\hat{s}_t$ rather than environment states $s_t$ produced by $E$ as happens during training. Following [9], to avoid performance degradation at inference time due to this mismatch, we propose to train the action module using as input to the dynamics network encoded environment states for the first $t$ steps and reconstructed environment states for the following ones. In all the experiments we use 4 initial encoded environment states.

In addition, we employ a curriculum learning strategy where we linearly increase the length of the reconstructed sequences during training. In particular, at the beginning of training, sequences of 5 environment states are reconstructed, while at the end of the annealing period at step 25.000 the reconstructed sequence length is set to 9. We use a batch size of 64 sequences.

Following [9], we set the number of actions $K$ to 7 for all the experiments.

We set $\lambda_{\text{rec}} = 1.0$, $\lambda_{\text{act}} = 0.15$, $\lambda_{\Delta} = 0.1$ and $\lambda_{\text{G}} = 0.1$ in the computation of the total loss term.

The action module is trained on 1x Nvidia RTX 8000.

**Loss masking.** During training of the action module, some frames may not contain a valid detection for each playable object. This may be due to the absence of the object in the frame or due to missed detections. We address this issue by computing the loss terms $\mathcal{L}_{\text{rec}}$, $\mathcal{L}_{\text{act}}$ and $\mathcal{L}_{\Delta}$ by taking into account only the prefix of the input sequence where the object has always been detected.

## 3. Datasets

In this section we describe the proposed *Minecraft* and *Tennis* datasets for the training of playable environments in in Sec. 3.1 and in Sec. 3.3, respectively. We also present details for the *Minecraft Camera* dataset in Sec. 3.2 and the *Static Tennis* dataset in Sec. 3.4.

## 3.1. Minecraft Dataset

We collect videos where two *Minecraft* players perform a sparring session. To acquire the sequences we build a Minecraft 16.4 [2] plugin based on [3] (GNU GPL v3). The plugin records Minecraft playing sessions and provides a GUI to replay the recorded sessions and render them under arbitrary camera trajectories. In addition, it is possible to re-render each sequence using different sets of textures for the environment and the players. Our plugin complements the rendered video with metadata containing camera intrinsic and extrinsic parameters and information about the entities that are present in the scene (eg. players). We release the code for the modified plugin to foster the use of Minecraft as a research tool.

We acquire sequences under varying illumination conditions where the camera slowly circles in a dome around the sparring area. Scenes at both day and night are recorded. The training set is composed of 69,600 frames in 1024x576 resolution at 20fps, divided in sequences of 400 frames. The test set and the validation set are composed each of 2,960 frames in sequences of 16 frames at 5fps. A total of 16 unique player identities are present. Note that the dataset can be automatically re-generated at arbitrary resolution and framerate.

## 3.2. Minecraft Camera Dataset

In order to evaluate the camera manipulation capabilities of our method, we collect additional *Minecraft* sequences. In each sequence, the camera starts from an initial pose and is then moved in the neighborhood of the original position in a circular pattern to synthesize novel views of the scene. To allow a better evaluation of novel view synthesis, the rendered scene remains unchanged in all the frames. We collect 24 sequences, each of length 16 frames for a total of 384 frames.

## 3.3. Tennis Dataset

We build a dataset of tennis match videos collected from YouTube. We collect a total of 43 tennis matches, featuring 86 different player appearances. The matches come from the following tournaments: 13 Us Open matches played on a concrete field, 7 Australian Open matches played on a concrete field, 8 Wimbledon matches played on grass, 15 Roland Garros matches played on red ground. In each original video, we extract portions of the match where the game is actively being played, from the moment of the service to the realization of the point, and discard parts where no action is occurring. We then process such raw sequences to extract camera calibration information and player bounding boxes.

| Tournament | Raw duration | Processed duration | Success |
|---|---|---|---|
| Us Open | 23.090 | 20.912 | 90.6 |
| Australian Open | 12.267 | 11.778 | 96.0 |
| Wimbledon | 12.014 | 11.574 | 96.3 |
| Roland Garros | 8.448 | 1.205 | 14.3 |
| Total | 55.819 | 45.469 | 81.5 |

Table 1. Statistics on the collected *Tennis* dataset indicating the duration of the collected raw sequences before camera calibration and player detection and the duration of the processed sequences for which camera calibration and player detection was performed successful. Success indicates the success rate in camera calibration and player detection. Dirt on the tennis lines causes a low success rate for Roland Garros sequences. Durations in seconds, success rate in %.

To perform camera calibration, we exploit the known geometry of the tennis field, using the field itself as a calibration pattern. To detect landmarks on the tennis field, we follow [4] using the implementation provided in [1] (BSD 3-Clause). Note that camera parameters obtained in this way present noise due to imprecision in the estimation of landmarks on the tennis field. We discard sequences with implausible camera poses or where camera calibration is excessively noisy using variance in the estimated camera position on the field as a noise estimator. In some sequences, camera calibration fails due to the inability to correctly detect the landmarks. This often happens in sequences from Roland Garros where the field lines are frequently covered by red ground and causes a high number of sequences from Roland Garros to be discarded. When possible, we provide camera parameters estimations for frames where landmark detection fails by interpolating the camera parameters of neighboring frames.

To perform player detection, we make use of a pretrained FasterRCNN [13] model. We employ the camera calibration parameters obtained in the preceding step to discard detections associated with people other than the players, *i.e.* referees, ball catchers and spectators. We notice that detection is sometimes inaccurate or fails due to the wide range of movements performed by the players. This happens more frequently for the player positioned further from the camera. Similarly to camera calibration, we discard sequences where detection of one of the players presents an excessive number of failures. When possible, in case of failed detections in one frame, we interpolate the detections of neighboring frames to produce an estimated detection.

The sequences where both camera calibration and player detection complete successfully are selected to be part of the dataset. Tab. 1 shows the duration statistics for the raw sequences before camera calibration and player detection and for the sequences that were successfully processed. A total of 12.6 hours of videos at 5 fps and 512x288 resolution are successfully processed and are organized in our dataset as follows: 212464 frames as training set organized in sequences of variable length, 6000 frames as validation set organized in sequences of 16 frames, and 6384 frames for test divided in sequences of 16 frames. Note that the original videos are in 1920x1080 resolution and 25fps, so the dataset can be automatically re-generated up to such resolution and framerate.

### 3.4. Static Tennis Dataset

We adopt the *Tennis* dataset of [9] to compare with previous playable video generation methods. The dataset is composed of video sequences from 2 matches recorded in the same arena and features 4 players with similar appearance. A total of 40 minutes of training videos is present. The validation and the test set are composed, respectively, of 2800 and 3280 frames divided in sequences of 16 frames acquired at 5fps. In order to satisfy the assumptions of previous methods, the dataset features limited camera movement and each video is cropped to depict only a single player in the lower part of the field.

## 4. Experiments

In this section we show additional experimental results. Our experiments are complemented by a website (see willimenapace.github.io/playable-environments-website) showing additional video qualitative results including video sequences produced interactively by users, camera and style manipulation results, and samples of dataset sequences. Sec. 4.1 shows a user study for the evaluation of video quality, Sec. 4.2 shows additional playability evaluation results, Sec. 4.3 describes the user study for the evaluation of playability, Sec. 4.4 shows ablation results for the synthesis module, Sec. 4.5 shows ablation results for the action module, Sec. 4.6 presents additional results for camera manipulation and Sec. 4.7 shows style manipulation results.

### 4.1. Synthesis Module User Study

In order to further evaluate video quality improvements with respect to CADDY [9], we run a user study on video quality against [9] on the *Static Tennis* dataset, the original dataset of [9]. Note that, for fairness of comparison, we make use of the *Static Tennis* dataset which does not feature the several challenges addressed by our method but not by CADDY [9], such as wide camera movements, multiple players and changes in appearance.

We create 206 video pairs with the two methods and ask 3 distinct AMT users to express their preference in terms of video quality between the two videos. 618 votes expressed by 18 distinct AMT users are gathered and assign a preference of 81.6% to our method.

| | *Aux.* | *H.Res.* | $\mathcal{L}_\Delta$ | LPIPS↓ | FID↓ | FVD↓ | Δ-*MSE*↓ | Δ-*Acc*↑ | ADD↓ | MDR↓ |
|---|---|---|---|---|---|---|---|---|---|---|
| (i) | | | | 0.743 | 288 | 3667 | 1.0 | (100) | **15.1** | 96.8 |
| (ii) | ✓ | | | 0.773 | 296 | 2906 | 0.999 | 52.8 | 56.3 | 93.2 |
| (iii) | ✓ | ✓ | | 0.677 | 211 | 2213 | 0.998 | 53.7 | 42.5 | 82.9 |
| (iv) | ✓ | | ✓ | 0.707 | 275 | 2553 | 0.467 | **83.3** | 56.2 | 92.2 |
| (v) | ✓ | ✓ | ✓ | 0.691 | 309 | 2187 | 0.439 | 82.6 | 17.4 | 95.1 |
| (Ours) | | | | **0.204** | **16.8** | **329** | **0.271** | 77.7 | 17.8 | **33.9** |

Table 2. Comparison with baselines on the *Minecraft* dataset. *Aux.*: use of auxiliary bounding box and camera pose information; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for Δ-*MSE*. Δ-*MSE*, Δ-*Acc* and MDR in %, ADD in pixels.

### 4.2. Playability Evaluation

In Tab. 2, we present playability evaluation results on the *Minecraft* dataset. As for the *Tennis* dataset, our method surpasses the baselines both in terms of video and action quality metrics. In particular, the high variation in camera pose in this dataset is not correctly modeled by the baseline methods which produce irrealistic results. Note that (i) and (iv) show a better Δ-*ACC* than our method which is explained by their learned action space which only discovers a reduced number of action categories as confirmed by the high Δ-*MSE* and by Fig. 5.

We show qualitatives results for both the *Minecraft* and *Tennis* datasets in Fig. 6 and Fig. 7, respectively. The movements of the players reconstructed by our method match the ones in the ground-truth sequence, indicating that a good action representation is learned. In addition, our model synthesizes players performing motions that are more realistic than the ones produced by the baselines.

We show a representation of the learned action space on both the *Minecraft* and *Tennis* datasets in Fig. 5.

### 4.3. Playability User Study

To further evaluate the quality of the action space we perform a user study (see Tab. 3) on the *Tennis* dataset, following the protocol of *Menapace et al.* [9]. We sample a set of 26 frames from the test set and for each frame we produce video continuations conditioned on each learned action. Two separate videos are produced for each initial frame and action, the first cropped on the lower tennis field, the second cropped on the upper tennis field to ensure a single tennis player is depicted in each video. For each produces sequence, we ask 3 *Amazon Mechanical Turk* users

| | *Aux.* | *H.Res.* | $\mathcal{L}_{\Delta}$ | Agreement↑ | Diversity↑ | *Other* votes |
|---|---|---|---|---|---|---|
| CADDY [9] (iii) | ✓ | ✓ | | 0.353 | **1.77** | 1.86 |
| CADDY [9] (v) | ✓ | ✓ | ✓ | 0.170 | 1.71 | 24.7 |
| (Ours) | | | | **0.444** | 1.7 | 0.80 |

Table 3. User study results on the *Tennis* dataset. *Aux.*: use of bounding boxes and camera pose; *H.Res.*: use of the high resolution model; $\mathcal{L}_{\Delta}$: use of the loss for $\Delta$-*MSE*. *Other* votes in %.

to choose which is the performed action between a set of options. We then measure agreement between the users using the Fleiss' kappa measure [5] and compute diversity in the generated actions using entropy of user-selected actions. We consider only the baselines with the highest video quality and with the best action space metrics. Our model achieves the best agreement and comparable diversity to the baselines indicating that the model generates videos that are consistently conditioned by the action and that no mode collapse of the learned actions is present.

## 4.4. Synthesis Module Ablation Study

We present ablation study results for the synthesis module on the *Tennis* dataset in Tab. 4 and present qualitative results in Fig. 3 and Fig. 4, respectively, for the *Minecraft* and *Tennis* datasets. The evaluation confirms the importance of the use of style modulation layers to model appearance changes and of the bending network to model deformations. In addition, without our feature renderer $F$, the model produces blurry results and lacks details such as the wrinkles on the clothes. Introducing the feature renderer $F$ allows the model to be trained on complete patches applying the perceptual loss term. This, in conjunction with the ability of ConvNets to model inter-pixel relationships, enables the model to reduce blur and to generate clothing that contains more detailed wrinkles. We also note that, in some sequences, the feature renderer is capable of generating realistic player shadows in portions of the image that lie outside of the bounding volume $\beta$ for the radiance field of the player. We ascribe this to the capacity of the ConvNet $F$ to model the correlation between the presence of the player and of its shadow.

## 4.5. Action Module Ablation Study

In Fig. 12 we show qualitative results for the action module ablation study on the *Minecraft* dataset. We note that, while method variations that do not use the temporal discriminator $D$ produce player movements closer to the ground-truth sequence, they tend to produce less realistic motions for the moving players.

| Var. | *Multi* ⟨**3**⟩ | $\pi$ ⟨**4**⟩ | $w$ ⟨**5**⟩ | $F$ ⟨**6**⟩ | LPIPS↓ | FID↓ | FVD↓ | ADD↓ | MDR↓ |
|---|---|---|---|---|---|---|---|---|---|
| (a) | | | | | 0.505 | 256.1 | 7011 | 72.2 | 85.6 |
| (b) | ✓ | | | | 0.619 | 271.4 | 10697 | 3.57 | 100.0 |
| (c) | ✓ | ✓ | | | 0.624 | 253.5 | 7280 | 2.65 | 28.6 |
| (d) | ✓ | ✓ | ∼ | | 0.319 | 96.3 | 3303 | 79.8 | 85.4 |
| (e) | ✓ | ✓ | ✓ | | 0.286 | 39.3 | 638 | 3.11 | 7.34 |
| (f) | ✓ | ✓ | ✓ | ∼ | 0.272 | 39.2 | 2678 | 46.9 | 60.5 |
| Full | ✓ | ✓ | ✓ | ✓ | **0.167** | **17.1** | **497** | **2.56** | **3.90** |

Table 4. Synthesis module ablation results on the *Tennis* dataset. *Multi*: use of multi-object modeling, $\pi$: use of deformation, $w$: use of style modulation layers or of direct style encoding (∼), $F$: use of the feature renderer or of the simplified renderer (∼). ADD in pixels, MDR in %.

| Var. | *Multi*⟨**3**⟩ | $\pi$⟨**4**⟩ | $w$⟨**5**⟩ | $F$⟨**6**⟩ |
|---|---|---|---|---|
| (a) | | | | |
| (b) | ✓ | | | |
| (c) | ✓ | ✓ | | |
| (d) | ✓ | ✓ | ∼ | |
| (e) | ✓ | ✓ | ✓ | |
| (f) | ✓ | ✓ | ✓ | ∼ |
| Full | ✓ | ✓ | ✓ | ✓ |
| Ground truth | | | | |

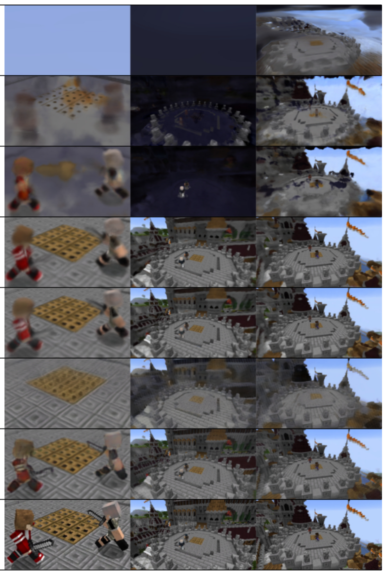


Figure 3. Synthesis module reconstruction results on the *Minecraft* dataset. The first image is cropped for better visualization. *Multi*: use of multi-object modeling, $\pi$: use of deformation, $w$: use of style modulation layers or of direct style encoding (∼), $F$: use of the feature renderer or of the simplified renderer (∼).

## 4.6. Camera Manipulation Evaluation

We evaluate camera manipulation capabilities for our method on the *Tennis* dataset. We consider each test sequence and sample two random camera poses by applying noise to the camera pose parameters in the first frame. We then generate a camera trajectory by interpolating between the two poses and render the sequence from the novel camera trajectory. Note that in the *Tennis* dataset the largest portion of the image is occupied by the field plane. Consequently, we can approximately match each image rendered from the novel trajectory to the corresponding original im-

| Var. | *Multi*⟨**3**⟩ | $\pi$⟨**4**⟩ | $w$⟨**5**⟩ | $F$⟨**6**⟩ |
|---|---|---|---|---|
| (a) | | | | |
| (b) | ✓ | | | |
| (c) | ✓ | ✓ | | |
| (d) | ✓ | ✓ | ∼ | |
| (e) | ✓ | ✓ | ✓ | |
| (f) | ✓ | ✓ | ✓ | ∼ |
| Full | ✓ | ✓ | ✓ | ✓ |
| Ground truth | | | | |

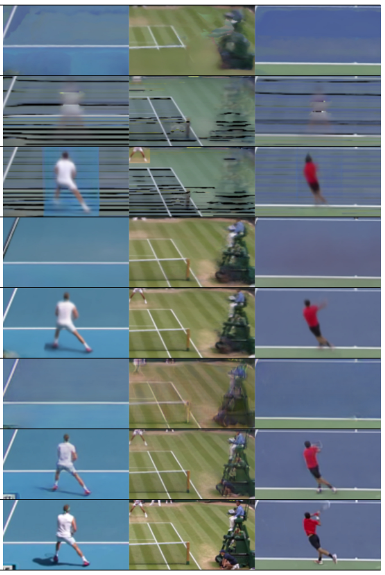


Figure 4. Synthesis module reconstruction results on the *Tennis* dataset. The image is cropped for better visualization. *Multi*: use of multi-object modeling, $\pi$: use of deformation, $w$: use of style modulation layers or of direct style encoding ($\sim$), $F$: use of the feature renderer or of the simplified renderer ($\sim$).

| | *Aux.* | *H.Res.* | $\mathcal{L}_{\Delta}$ | LPIPS↓ | FID↓ | ADD↓ | MDR↓ |
|---|---|---|---|---|---|---|---|
| (i) | | | | 0.659 | 224 | 50.4 | 82.0 |
| (ii) | ✓ | | | 0.623 | 170 | 54.7 | 33.7 |
| (iii) | ✓ | ✓ | | 0.425 | 26.1 | 38.1 | 29.2 |
| (iv) | ✓ | | ✓ | 0.634 | 164 | 57.4 | 41.7 |
| (v) | ✓ | ✓ | ✓ | 0.574 | 216 | 35.0 | 66.7 |
| (Ours) | | | | **0.205** | **20.5** | **13.9** | **5.76** |

Table 5. Camera control evaluation results on the *Tennis* dataset. *Aux.*: use of bounding boxes and camera pose; *H.Res.*: use of the high resolution model; $\mathcal{L}_{\Delta}$: use of the loss for $\Delta$-*MSE*. ADD in pixels, MDR in %.

age by applying a homography. We then compute reconstruction losses between the original and the generated sequence warped according to the corresponding homography. While this evaluation does not account for parts of the image that do not lie on the field plane such as the players, it can be used to detect failure cases. We show the results in Tab. 5. LPIPS, ADD and MDR highlight that our method obtains better consistency than the baselines in the generation of novel camera views. In addition, we show qualitatives in Fig. 8 and Fig. 9, respectively, for the *Minecraft* and *Tennis* dataset.

## 4.7. Style Manipulation Evaluation

In Fig. 11 and Fig. 10 we show qualitative style manipulation results obtained with our method. We consider a target style image from which we extract the style code $w$ for each object. We then replace the style code in the environment state extracted from a source image with the target style and re-render the image using the synthesis module. Our method can successfully change the appearance of the players and of the static scene elements.

## 5. Discussion

The following section discusses the main limitations of the method and social implications.

**Limitations.** Our method performs some assumptions on the structure of the environment. First, while the appearance of the environment can change, the method assumes a constant geometry of the environment, preventing it from operating on datasets with actions performed in environments with different geometries. Second, in order to estimate the pose of dynamic objects, the method assumes that moving objects are located on the $y = 0$ plane. Lastly, while on the *Minecraft* dataset our method is learns a full 3D environment, on the *Tennis* dataset the presence of camera rotations but not of camera translations makes the problem of recovering the field geometry ill-posed. We thus impose a “flat world” prior in order to regularize the geometry learned on this dataset, which is detailed in Sec. 1.4. This prior allows wider camera manipulations at inference time, but has the effect of projecting parts of the background on flat surfaces, causing visual artifacts when the position of the camera differs from the one of the original dataset.

We also note that our method exhibits blur artifacts or missing-part artifacts in regions of the scene corresponding to thin and fast-moving objects such as the limbs of players or the rackets. The challenging nature of the *Tennis* dataset with frequent motion blur, object parts as small as a few pixels (eg. limbs) and noise in camera calibration is the main reason behind these artifacts that are partially addressed by our feature renderer.

When learning actions, it is important to consider both ease of manipulation, and the portion of interesting actions that can be captured. Users are typically mainly interested in manipulating the position of objects which is often correlated to the change in poses. Thus, to ease the manipulation, we found an ideal solution in learning actions related to changes in position and letting the model synthesize the corresponding change in poses. The main limitation caused by this tradeoff between action space simplicity and action space expressiveness is that it is difficult to control the synthesis of actions featuring changes in pose that are not di-

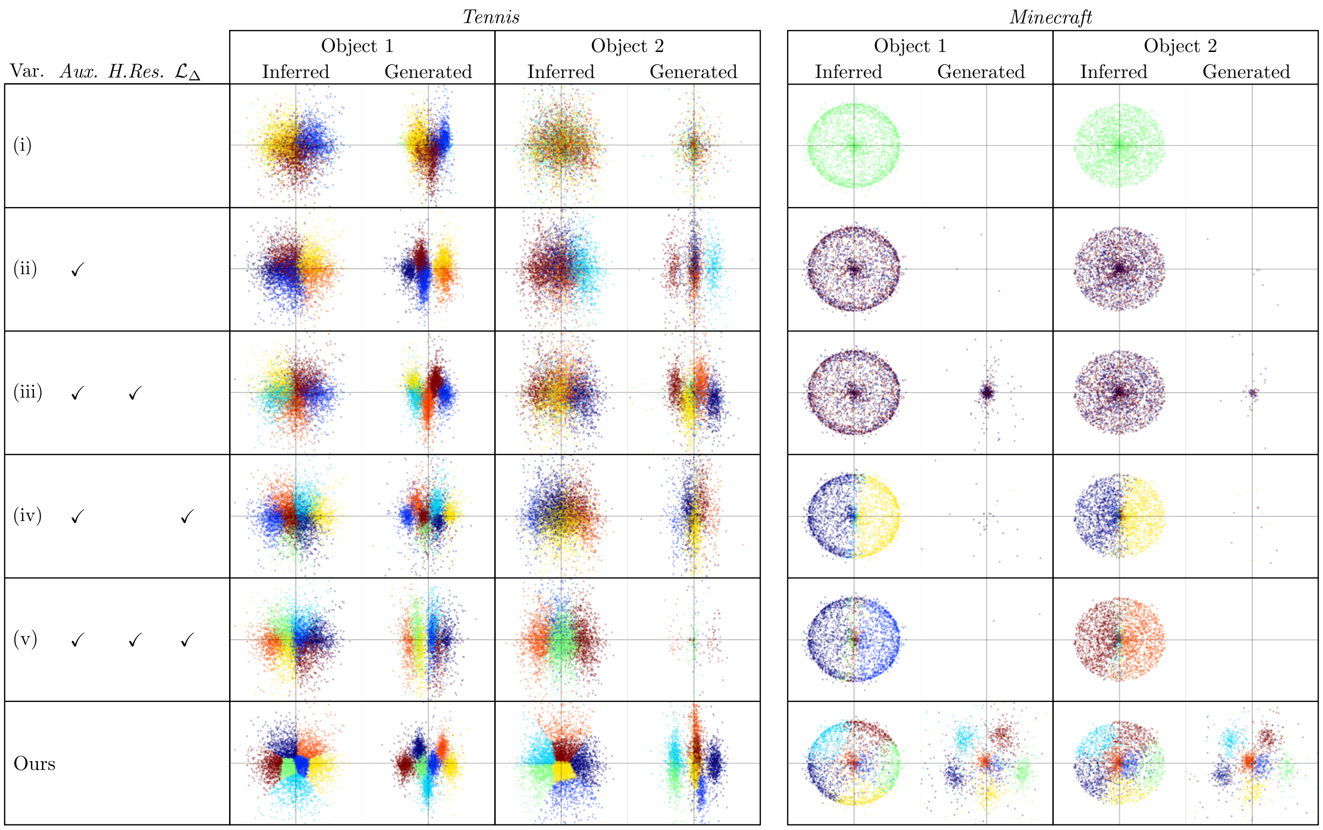


Figure 5. Visualization of the learned action space on the *Minecraft* and *Tennis* datasets for our method and the baselines. Each plot shows $\Delta$ movements of the playable objects measured on the ground plane between a pair of successive frames. Colors indicate the different action label that is associated to each movement. 'Inferred' movements are measured on the test sequences, thus the plots shows the actions inferred by the action network as a function of the input states. 'Generated' movements are measured on the generated sequences, thus the plots show the movement generated by the dynamics network as a response to the current action input. It can be observed that our method produces sharper decision boundaries between the actions associated to movement. In addition, the distribution of 'Generated' movements produced as a response to each action input matches the distribution of the 'Inferred' movements associated to the same action. *Aux.*: use of auxiliary bounding box and camera pose information; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for $\Delta$-*MSE*.

rectly correlated to changes in position, such as the swing of the racket.

Lastly, we remark that at inference time objects in the scene are animated independently from each other. This can generate artifacts such as both tennis players swinging the rackets at the same time and makes it not possible to capture interactions between objects in the scene.

**Social implications.** Similarly to methods operating on face and human appearances, our method could be used to deceive, and can potentially be used to tamper video sequences for nefarious purposes. However, we note that the constant environment geometry assumed by the method provides some protection against fraudulent uses since the method requires a certain scene to appear in many videos in the training dataset in order to allow its subsequent manipulation. This would make it not straightforward to tamper an isolated video for which the malicious individual does not dispose of a corresponding collection of videos of the same scene to use for training. We believe that the benefits brought in terms of novel creative applications and enhancements to user creativity outweigh the potential risks.

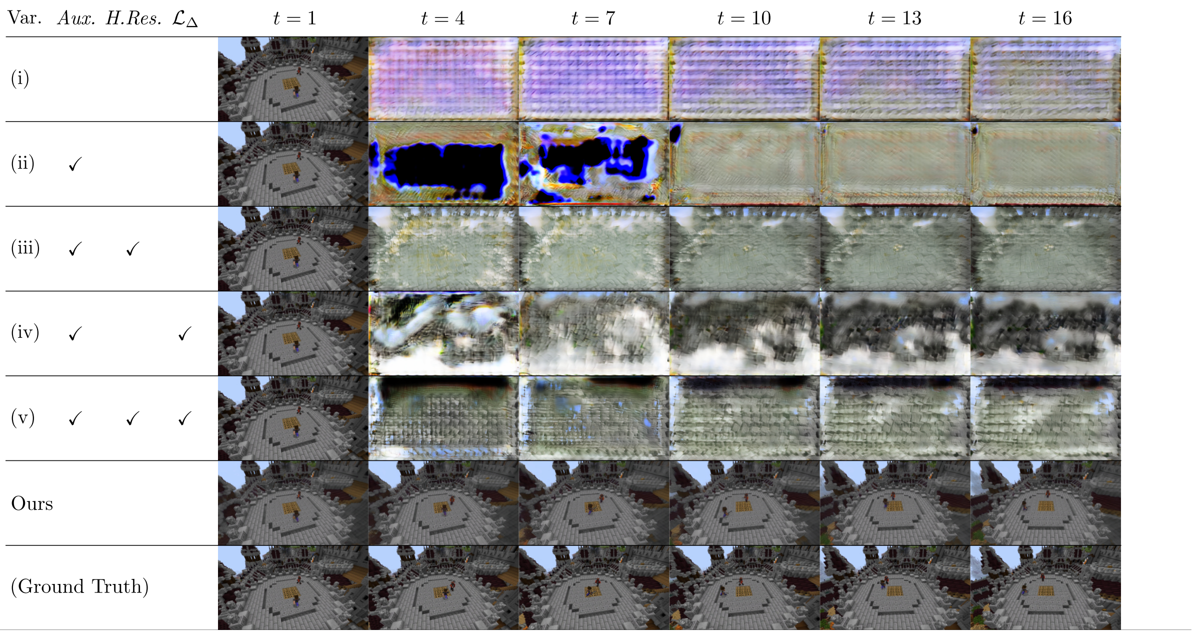

Var. *Aux.* *H.Res.* $\mathcal{L}_\Delta$
$t = 1$ $t = 4$ $t = 7$ $t = 10$ $t = 13$ $t = 16$
(i)
(ii) ✓
(iii) ✓ ✓
(iv) ✓ ✓
(v) ✓ ✓ ✓
Ours
(Ground Truth)

Figure 6. Reconstruction results on the *Minecraft* dataset. Starting from the first frame, the dynamics network reconstructs the ground truth video using the sequence of discrete actions inferred by the action network on the original sequence. *Aux.*: use of auxiliary bounding box and camera pose information; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for $\Delta$-*MSE*.

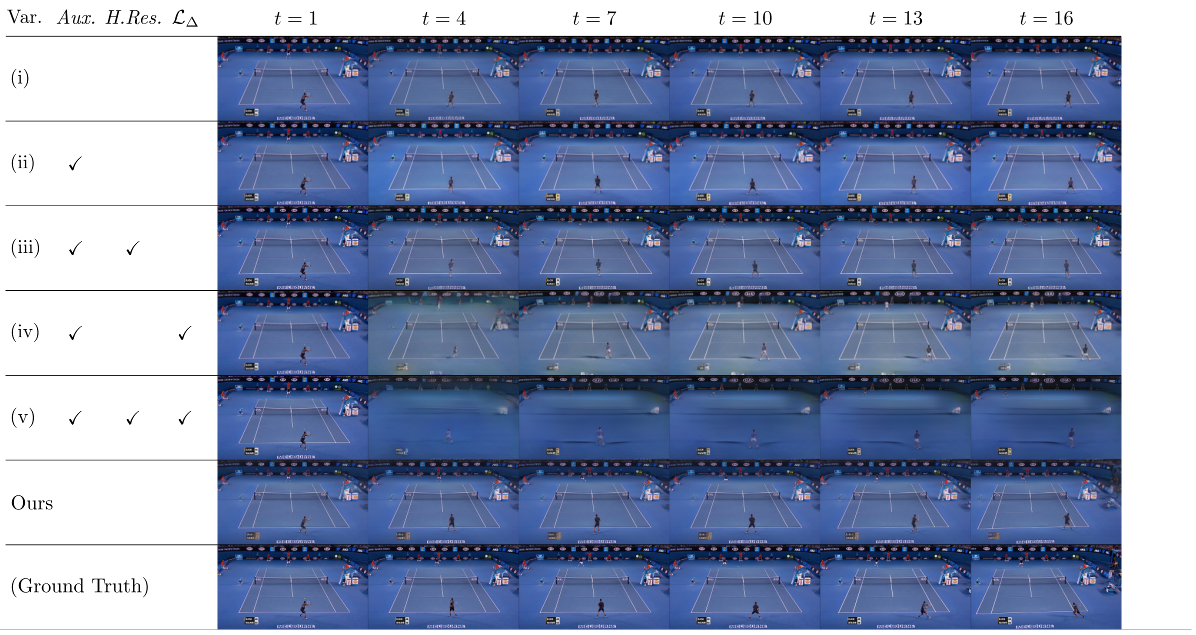

Var. *Aux.* *H.Res.* $\mathcal{L}_\Delta$
$t = 1$ $t = 4$ $t = 7$ $t = 10$ $t = 13$ $t = 16$
(i)
(ii) ✓
(iii) ✓ ✓
(iv) ✓ ✓
(v) ✓ ✓ ✓
Ours
(Ground Truth)

Figure 7. Reconstruction results on the *Tennis* dataset. Starting from the first frame, the dynamics network reconstructs the ground truth video using the sequence of discrete actions inferred by the action network on the original sequence. *Aux.*: use of auxiliary bounding box and camera pose information; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for $\Delta$-*MSE*.

| Var. | *Aux.* | *H.Res.* | $\mathcal{L}_\Delta$ |
|---|---|---|---|
| (i) | | | |
| (ii) | ✓ | | |
| (iii) | ✓ | ✓ | |
| (iv) | ✓ | | ✓ |
| (v) | ✓ | ✓ | ✓ |
| Ours | | | |
| (Ground Truth) | | | |

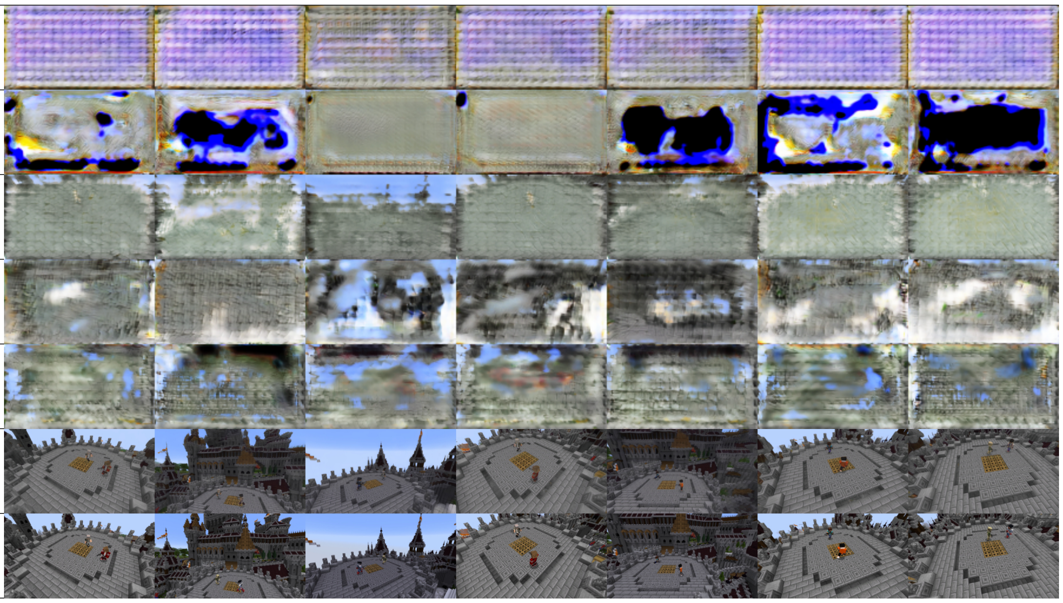


Figure 8. Camera manipulation results on the *Minecraft Camera* dataset. Ground truth shows the reference frame, upper rows show camera manipulation results. *Aux.*: use of auxiliary bounding box and camera pose information; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for $\Delta$-*MSE*.

| Var. | *Aux.* | *H.Res.* | $\mathcal{L}_\Delta$ |
|---|---|---|---|
| (i) | | | |
| (ii) | ✓ | | |
| (iii) | ✓ | ✓ | |
| (iv) | ✓ | | ✓ |
| (v) | ✓ | ✓ | ✓ |
| Ours | | | |
| (Ground Truth) | | | |

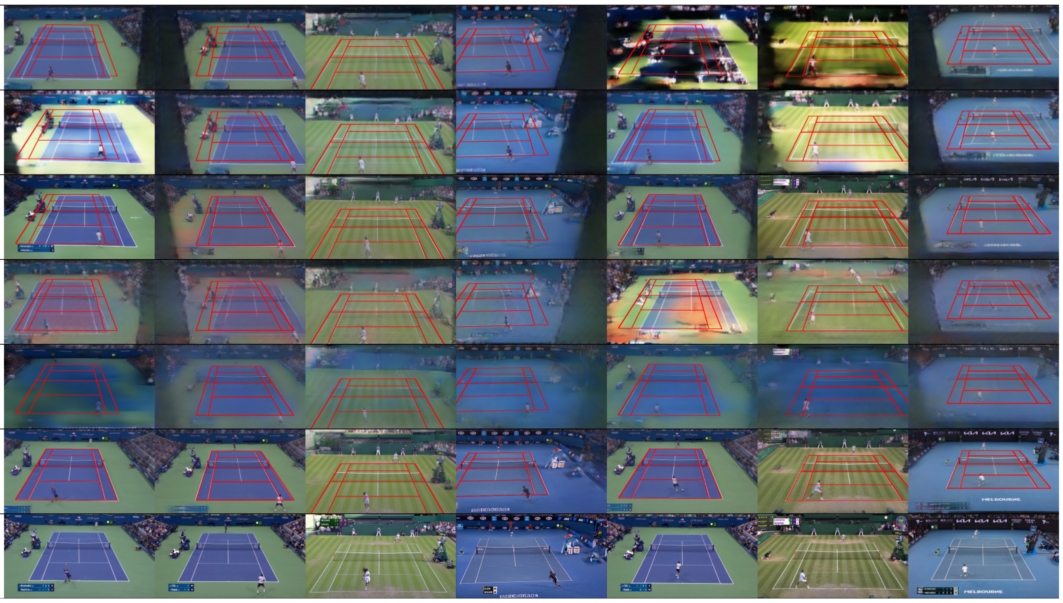


Figure 9. Camera manipulation results on the *Tennis* dataset. Ground truth shows the reference frame, upper rows show camera manipulation results. To highlight camera manipulation errors, we overlay each generated image with the position where the field lines should be synthesized under the manipulated camera pose. *Aux.*: use of auxiliary bounding box and camera pose information; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for $\Delta$-*MSE*.

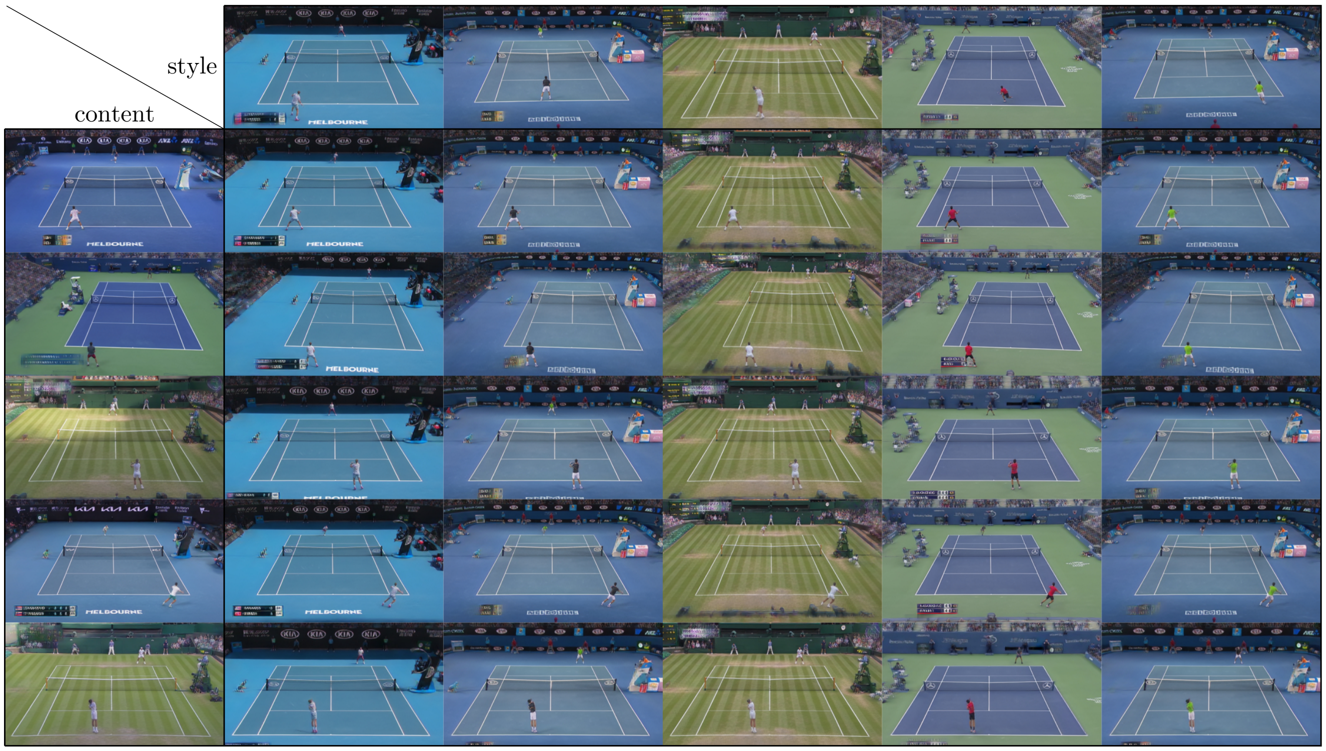


Figure 10. Style transfer results on the *Tennis* dataset. The topmost row shows the target style, the leftmost column shows the image in which the style is injected.

## References

[1] Implementation of: Robust camera calibration for sport videos using court models. https://github.com/gchlebus/tennis-court-detection. Accessed: 2021-11-12. 5

[2] Minecraft. https://www.minecraft.net. Accessed: 2021-11-12. 5

[3] Replaymod. https://github.com/ReplayMod/ReplayMod. Accessed: 2021-11-12. 5

[4] Dirk Farin, Susanne Krabbe, Peter H. N. de With, and Wolfgang Effelsberg. Robust camera calibration for sport videos using court models. In *IS&T/SPIE Electronic Imaging*, 2003. 5

[5] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971. 7

[6] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2017. 2, 3

[7] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proceedings of the European Conference of Computer Vision (ECCV)*, 2016. 2

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 4

[9] Willi Menapace, Stephane Lathuiliere, Sergey Tulyakov, Aliaksandr Siarohin, and Elisa Ricci. Playable video generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10061–10070, 2021. 2, 4, 6, 7

[10] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference of Computer Vision (ECCV)*, 2020. 1, 2

[11] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018. 4

[12] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021. 2

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 91–99, 2015. 5

[14] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020. 3

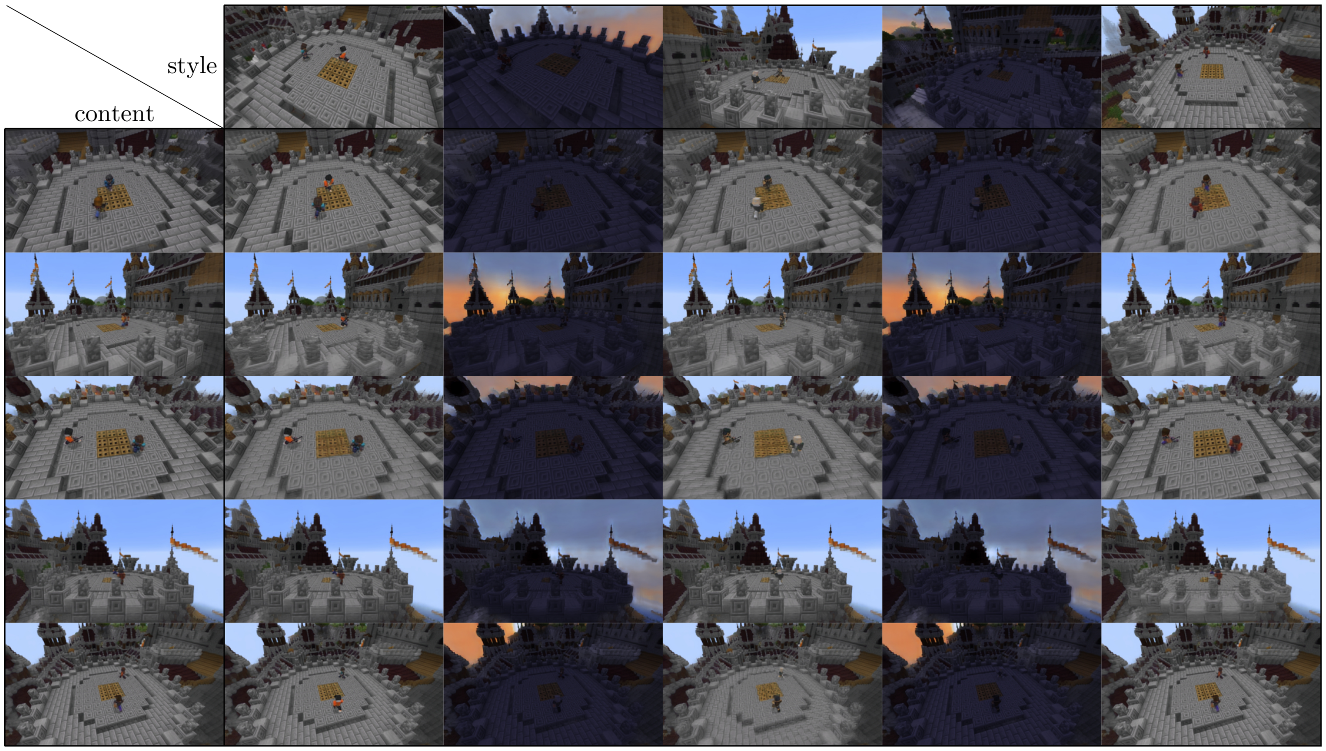


Figure 11. Style transfer results on the *Minecraft* dataset. The topmost row shows the target style, the leftmost column shows the image in which the style is injected.

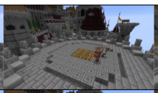
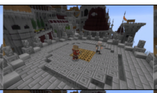
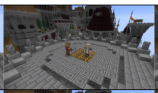
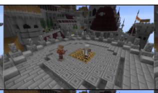
| Var. | *Rel.* | $D$ | $\mathcal{L}_\Delta$ | $t = 1$ | $t = 4$ | $t = 7$ | $t = 10$ | $t = 13$ | $t = 16$ |
|---|---|---|---|---|---|---|---|---|---|
| (A) | | | | |  |  |  |  | |
| (B) | ✓ | | | | | | | | |
| (C) | ✓ | | ✓ | | | | | | |
| (D) | ✓ | ✓ | | | | | | | |
| (Full) | ✓ | ✓ | ✓ | | | | | | |
| (Ground Truth) | | | | | | | | | |

Figure 12. Reconstruction results on the *Minecraft* dataset. Starting from the first frame, the dynamics network reconstructs the ground truth video using the sequence of discrete actions inferred by the action network on the original sequence. *Rel.*: use of camera relative residual $\Delta$ output, $D$: use of the temporal discriminator, $\mathcal{L}_\Delta$: use of the loss for $\Delta$-*MSE*.