

Plotting Behind the Scenes: Towards Learnable Game Engines

WILLI MENAPACE*, University of Trento, Italy

ALIAKSANDR SIAROHIN, Snap Inc., USA

STÉPHANE LATHUILIÈRE, LTCI, Télécom Paris, Institut Polytechnique de Paris, France

PANOS ACHLIOPTAS, Snap Inc., USA

VLADISLAV GOLYANIK, MPI for Informatics, SIC, Germany

ELISA RICCI, University of Trento, Fondazione Bruno Kessler, Italy

SERGEY TULYAKOV, Snap Inc., USA

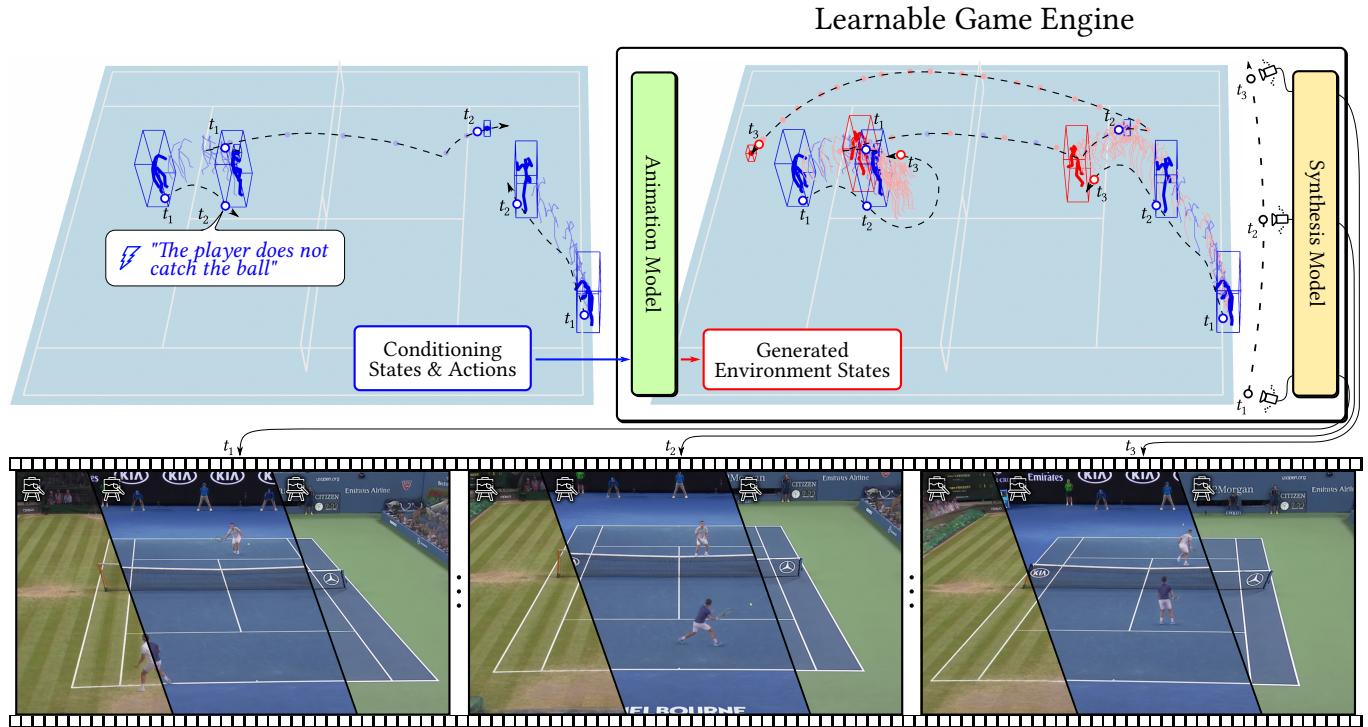


Fig. 1. We propose Learnable Game Engines (LGEs), a game-engine-like model that is learned from annotated videos. LGEs enable the generation of videos using a wide spectrum of [conditioning signals](#) such as player poses, object locations, and fine-grained textual actions (see \checkmark) indicating what each player should do. Our *Animation Model* uses this information to generate future, past, or interpolated [environment states](#) according to the learned game logic and laws of physics. At this stage, the model is able to perform complex action reasoning such as generating a winning shot if the action “the [other] player does not catch the ball” is specified, as shown in the figure. To accomplish this goal, the model decides that the bottom player should hit the ball with a “lob” shot, sending the ball high above the opponent, who is unable to catch it. As a game engine, LGEs render the scene from a user-defined viewpoint (see \Rightarrow) using a *Synthesis Model* where the style of the scene (see \P) can be controlled explicitly. The complete video and additional examples are shown in the [Supp. Website](#).

Game engines are powerful tools in computer graphics. Their power comes at the immense cost of their development. In this work, we present a framework to train game-engine-like neural models, solely from monocular annotated

*Work performed while the author was an intern at Snap Inc.

Authors’ addresses: Willi Menapace, University of Trento, Italy, willi.menapace@unitn.it; Aliaksandr Siarohin, Snap Inc., USA, asiarohin@snapchat.com; Stéphane Lathuilière, LTCI, Télécom Paris, Institut Polytechnique de Paris, France, stephane.lathuiliere@telecom-paris.fr; Panos Achlioptas, Snap Inc., USA, pachlioptas@gmail.com; Vladislav Golyanik, MPI for Informatics, SIC, Germany, golyanik@mpi-inf.mpg.de; Elisa Ricci, University of Trento, Fondazione Bruno Kessler, Italy, e.ricci@unitn.it; Sergey Tulyakov, Snap Inc., USA, stulyakov@snapchat.com.

videos. The result—a Learnable Game Engine (LGE)—maintains states of the scene, objects and agents in it, and enables rendering the environment from a controllable viewpoint. Similarly to a game engine, it models the logic of the game and the underlying rules of physics, to make it possible for a user to *play* the game by specifying both high- and low-level action sequences. Most captivatingly, our LGE unlocks the *director’s mode*, where the game is played by *plotting behind the scenes*, specifying high-level actions and goals for the agents in the form of *language* and *desired states*. This requires learning “game AI”, encapsulated by our animation model, to navigate the scene using high-level constraints, play against an adversary, devise the strategy to win a point. The key to learning such game AI is the exploitation of a large and

diverse text corpus, collected in this work, describing detailed actions in a game and used to train our animation model. To render the resulting state of the environment and its agents, we use a compositional NeRF representation used in our synthesis model. To foster future research, we present newly collected, annotated and calibrated large-scale Tennis and Minecraft datasets. Our method significantly outperforms existing neural video game simulators in terms of rendering quality. Besides, our LGEs unlock applications beyond capabilities of the current state of the art. Our framework, data, and models are available at learnable-game-engines.github.io/lge-website.

CCS Concepts: • Computing methodologies → Rendering; Animation.

Additional Key Words and Phrases: neural radiance fields, diffusion models, human motion generation, language modeling

1 INTRODUCTION

Advancements in graphics brought new capabilities to game engines. Their primary purpose was to democratize game development but due to the supported features and quality, their impact quickly reached a variety of creative applications spanning AR, VR, data generation [Cao et al. 2020; Hoffman et al. 2018], and, most recently, virtual film production¹. Building the vast software ecosystem of a game engine is an enormously challenging task and, despite the sophistication reached by such systems, making video games still requires specialized equipment, acquisition of expensive data and 3D assets, and labor of trained experts. There are, however, thousands of videos with games already played and real-world matches spectated. *Would it be possible to learn a game engine using this data?*

Many learnable methods focus on reducing the need for manual labor and 3D assets in computer graphics [Holden et al. 2017; Kuang et al. 2022; Liu et al. 2021; Starke et al. 2019, 2020], but only provide narrow video game functions. More related to our work, neural video game simulation methods show that annotated videos can be used to learn to generate videos interactively [Davtyan and Favaro 2022; Huang et al. 2022; Kim et al. 2021, 2020; Menapace et al. 2021] and build 3D environments where agents can be controlled through a set of discrete actions [Menapace et al. 2022]. While bringing us closer to *learnable game engines*, when applied to complex or real-world environments, these works have several limitations: do not accurately model game logic, do not model physical interactions of objects in 3D space, do not learn fine-grained controls, do not allow for high-level goal-driven control of the game flow, and, finally, do not model game AI.

In this work, we present a framework for building game-engine-like models by observing a handful of annotated videos. Our framework supports a complete set of core game engine functions including rendering from a controllable viewpoint, modeling of game logic and physics, fine-grained character control, and game AI. Due to the versatility of the supported applications (see Sec. 4), we call our framework Learnable Game Engines (LGes).

To overcome the limitations of [Davtyan and Favaro 2022; Huang et al. 2022; Kim et al. 2021, 2020; Menapace et al. 2021, 2022], not only we model the *states* of an environment, but we also consider detailed textual representations of the actions taking place in it. We argue that training on user commentaries describing detailed actions

of a game greatly facilitates learning the game’s logic and game AI—important parts of our LGEs—and that such commentaries are a key component in enabling a series of important model capabilities related to fine-grained character control and high-level goal-driven control of the game flow.

In its simplest form, for games like tennis, this enables controlling each player in a fine-grained manner with instructions such as “*hit the ball with a backhand and send it to the right service box*”. Moreover, language enables users to take the *director’s mode* and perform high-level game-specific scenarios or scripts, specified by means of *natural language* and *desired states of the environment*. As an example, given desired starting and ending states, our learned game engines can devise in-between scenarios that led to the observed outcome. Most interestingly, as shown in Fig. 1, given the initial states of a real tennis video in which a player lost a point, our LGE prompted by the command “*the [other] player does not catch the ball*” can perform the necessary action to win the point.

Broadly speaking, a game maintains states of its environments [Curtis et al. 2022; Stanton et al. 2016; Starke et al. 2019], renders them using a controllable camera, and evolves them according to user commands, actions of non-playable characters controlled by the game AI, and the game’s logic. Our Learnable Game Engines follow this high-level structure highlighted in Fig. 1. Our synthesis model maintains a state for every object and agent included in the game and renders them in the image space using the compositional NeRF [Mildenhall et al. 2020] of [Menapace et al. 2022] followed by a learnable enhancer for superior rendering quality. To model the logic of games and game AI that determine the evolution of the environment states, we introduce an animation model. Specifically, inspired by [Han et al. 2022], we train a *non-autoregressive text-conditioned* diffusion model which leverages masked sequence modeling to enable the fine-grained conditioning capabilities on which the above-mentioned applications are based. In particular, we show that using text labels describing actions happening in a game is instrumental in learning such capabilities. While certain prior work [Kim et al. 2021, 2020; Menapace et al. 2021, 2022] explored maintaining and rendering states of games, we are not aware of any generative method that attempts enabling fine-grained control, modeling sophisticated goal-driven game logic, and learning game AI to the extent explored in this paper.

The task of playing games and manipulating videos in the *director’s mode* has not been previously introduced in the literature. With this work, we attempt to introduce the task and set up a solid framework for future research. To do that, first and crucially, we collected two monocular video datasets. The first one is the Minecraft dataset containing 1.2 hours of videos, depicting a player moving in a complex environment. The second one is a large-scale real-world dataset with 15.5 hours of high-resolution professional tennis matches. For each frame in these datasets, we provide accurate camera calibration, 3D player poses, ball localization and, most importantly, diverse and rich text descriptions of the actions performed by each player in each frame.

In summary, our work brings the following contributions:

- A novel framework to build Learnable Game Engines from annotated monocular videos. It supports detailed rendering

¹Unreal and Unity engines are used to photorealistically render environments for film production.

of high-resolution, high-frame rate videos of scenes with articulated objects from arbitrary viewpoints. It can generate fine-grained actions specified by text prompts, model opponents, and perform goal-driven generation of complex action sequences. As far as we are aware, no existing work provides this set of capabilities under comparable data assumptions.

- A synthesis model, based on a compositional NeRF, producing videos at the original frame rate and doubling the output resolution with respect to [Menapace et al. 2022].
- An animation model, based on a text-conditioned diffusion model with a masked training procedure, which is key to support complex game logic, object interactions, game AI, and understanding fine-grained actions. It unlocks applications currently out of reach of state-of-the-art neural video game simulators (see Sec. 4).
- A large-scale 15h Tennis and a 1h Minecraft video datasets with camera calibration, 3D player poses, 3D ball localization, and detailed text captions.

2 RELATED WORK

2.1 Neural video game simulation

In the last few years, video game simulation using deep neural networks has emerged as a new research trend [Davtyan and Favaro 2022; Huang et al. 2022; Kim et al. 2021, 2020; Menapace et al. 2021, 2022]. The objective is to train a neural network to synthesize videos based on sequences of actions provided at every time step.

This problem was first addressed using training videos annotated with the corresponding action labels at each time step [Chiappa et al. 2017; Kim et al. 2020; Oh et al. 2015]. They consider a discrete action representation that is difficult to define a priori for real-world environments. More recently, [Kim et al. 2021] proposed a framework that uses a continuous action representation to model real-world driving scenarios. Devising a good continuous action representation for an environment, however, is complex. To avoid this complexity, [Menapace et al. 2021, 2022] propose to learn a discrete action representation. [Huang et al. 2022] expands on this idea by modeling actions as a learned set of geometric transformations, while [Davtyan and Favaro 2022] represents actions by separating them into a global shift component and a local discrete action component.

Among these works, *Playable environments* [Menapace et al. 2022] is the most closely related to ours. Rather than employing a 2D model, they use a NeRF-based renderer [Mildenhall et al. 2020] that enables them to represent complex 3D scenes. However, the employed discrete action representation shows limitations in complex scenarios such as tennis, where it is only able to capture the main movement directions of the players and does not model actions such as ball hitting. In contrast, we employ a text action representation that specifies actions at a fine level of granularity (i.e. which particular ball-hitting action is being performed and where the ball is sent), while remaining interpretable and intuitive for the user.

Besides, previous works perform generation in an autoregressive manner, conditioned on the actions and, therefore, are unable to perform constraint- or goal-driven generation for which non-sequential conditioning is necessary. We find the proposed text-based action representation to be crucial to unlock such applications.

2.2 Sequential data generation with diffusion models

In prior work, sequential data generation was mainly addressed with auto-regressive formulations combined with adversarial [Kwon and Park 2019] or variational [Babaeizadeh et al. 2018; Fortuin et al. 2020] generative models. Recently, diffusion models have emerged as a promising solution to this problem leading to impressive results in multiple applications such as audio [Chen et al. 2021; Kong et al. 2020; Lam et al. 2022; Leng et al. 2022] and video synthesis [Ho et al. 2022a,b; Singer et al. 2022], language modeling [Dieleman et al. 2022], and human motion synthesis [Dabral et al. 2022; Zhang et al. 2022]. Following this methodological direction [Tashiro et al. 2021], introduces a score-based diffusion model for imputing missing values in time series. They introduce a training procedure based on masks that simulates missing data. This approach motivates our choice of a similar masking strategy to generate the unknown environment states. In this work, we show that mask-based training is highly effective in modeling geometric properties together with textual data modalities.

2.3 Text-based generation

In recent years, we have witnessed the emergence of works on the problem of text-based generation. Several works address the problem of generating images [Ramesh et al. 2022, 2021; Rombach et al. 2021; Saharia et al. 2022] and videos with arbitrary content [Ho et al. 2022a,b; Hong et al. 2022; Singer et al. 2022], and arbitrary 3D shapes [Achlioptas et al. 2023; Jain et al. 2022; Lin et al. 2022; Poole et al. 2022].

Han *et al.* [Han et al. 2022] introduced a video generation framework that can incorporate various conditioning modalities in addition to text, such as segmentation masks or partially occluded images. Their approach employs a frozen RoBERTa [Liu et al. 2020] language model and a sequence masking technique. Fu *et al.* [Fu et al. 2022] propose an analogous framework. Our animation framework employs a similar masking strategy, but we model text conditioning at each timestep in the sequence, use diffusion models which operate on continuous rather than discrete data, and generate scenes that can be rendered from arbitrary viewpoints.

More relevant to our work, several papers introduced models to generate human motion sequences from text [Athanasius et al. 2022; Tevet et al. 2022]. Recently, diffusion models have shown strong performance on this task [Dabral et al. 2022; Zhang et al. 2022]. In these works, sequences of human poses are generated by a diffusion model conditioned on the output of a frozen CLIP text encoder. It is worth noting that these prior works model only a single human, while our framework supports multiple human agents and objects, and models their interactions with the environment.

2.4 Character Animation

Character animation is a long-standing problem in computer graphics. Several recent methods have been proposed that produce high-quality animations. Holden *et al.* [Holden et al. 2020] propose a learnable version of Motion Matching [Büttner and Clavet 2015] that formulates character animation as retrieval of the closest motion from a motion database and supports interaction with other

characters or objects. Other approaches model the evolution of characters using time series models conditioned on the preceding state and control signals [Holden et al. 2017; Lee et al. 2018; Starke et al. 2019, 2020]. Starke *et al.* [Starke et al. 2019] propose a model based on a mixture of experts that controls character locomotion and object interactions, in a follow-up work [Starke et al. 2020] they introduce local motion phases to model complex character motions and interaction with a second character.

To produce high-quality animations the methods rely on difficult-to-acquire motion capture data enriched with contact information [Holden et al. 2020; Starke et al. 2019, 2020], motion phases [Starke et al. 2019] or engineered action labels [Starke et al. 2019, 2020]. Additionally, handcrafted dataset-specific feature representations and mappings from user controls to such representations are often leveraged, and additional knowledge is injected through postprocessing steps such as inverse kinematics or external physics models. While these assumptions promote high-quality outputs, they come at a significant effort. In contrast, our method sidesteps these requirement by not using motion capture and basing user control on natural language that is cheaper to acquire (see Appx. I.1) and does not require manual engineering. Finally, character animation methods support limited goal-driven control such as interacting with a specific object while avoiding collisions [Starke et al. 2019]. In contrast, our method models complex game AI tasks such as environment navigation and modeling strategies to defeat the opponent, which are instrumental in learning a game engine.

2.5 Neural Rendering

Neural rendering was recently revolutionized by the advent of NeRF [Mildenhall et al. 2020]. Several modifications of the NeRF framework were proposed to model deformable objects [Li et al. 2022; Park et al. 2021a,b; Treitschke et al. 2021; Weng et al. 2022], and decomposed scene representations [Kundu et al. 2022; Menapace et al. 2022; Müller et al. 2022; Niemeyer and Geiger 2021; Ost et al. 2021; Wu et al. 2022]. In addition, several works improved the efficiency of the original MLP representation of the radiance field [Mildenhall et al. 2020] by employing octrees [Yu et al. 2021], voxel grids [Fridovich-Keil et al. 2022], triplanes [Chan et al. 2022], hash tables [Müller et al. 2022], or factorized representations [Chen et al. 2022].

Our framework is most related to that of [Weng et al. 2022], since we model player deformations using an articulated 3D prior and linear blend skinning (LBS) [Lewis et al. 2000]. Differently to them, however, we consider scenes with multiple players and apply our method to articulated objects with varied structures for their kinematic trees. While similar to the rendering framework of [Menapace et al. 2022], our framework does not adopt computationally-inefficient MLP representations, using voxel [Fridovich-Keil et al. 2022] or plane representations instead.

3 METHOD

This section introduces the *Learnable Game Engines* framework that allows the user to perform a range of dynamic scene editing tasks.

Similarly to traditional game engines that maintain states of each object, render the environment using a graphics pipeline, and have a model of game logic, we divide our LGE into two modules: a *synthesis*

model and an *animation model*. The task of the synthesis model is to generate an image given the representation of the environment state. The animation model, instead, aims at modeling the game’s logic, with player actions and interactions, in the high-level space of the environment states. Actions are modeled as text, which is an expressive, yet intuitive form of control for a wide range of tasks. The overview of our framework is provided in Fig. 2a.

In more detail, our LGE defines the state of the entire environment as the combination of all individual object states. Consequently, each individual state is the set of the object properties such as the position of each object in the scene, their appearance, or their pose. Formally, the environment state at time t can be represented by $s_t \in \mathbb{S} = (\mathbb{R}^{n_1} \times \dots \times \mathbb{R}^{n_P})$, P properties of variable length n_i defined as the union of the properties of each object. This state representation captures all variable aspects of each individual object in the environment, thus it can be used by the synthesis model to generate the scene.

On the other hand, the animation model predicts the evolution of an environment in time, which is represented by the sequence of its states $\{s_1, s_2, \dots, s_T\} = s \in \mathbb{S}^T$, where T is the length of the sequence. The LGE provides control over sequence generation with the help of user-defined conditioning signals that can take two forms: explicit state manipulation and high-level text-based editing. With respect to the former, the user could change the position of the tennis ball at time step t , and the model would automatically adapt the position of the ball in other nearby states. As far as the latter is concerned, users could provide high-level text-based values of actions such as “*The player takes several steps to the right and hits the ball with a backhand*” and the model would generate the corresponding sequence of states (see Fig. 3). These generic actions in the form of text are central to enabling high-level, yet fine-grained control over the evolution of the environment. To train our framework we assume a dataset of camera-calibrated videos, where each video frame is annotated with the corresponding states s and actions a .

3.1 Synthesis Model

In this section, we describe the synthesis model that renders states from arbitrary viewpoints (see Fig. 2c). We build our model based on a compositional NeRF [Menapace et al. 2022] framework which enables explicit control over the camera and represents a scene as a composition of different, independent objects. Thanks to the independent representation of objects, each object property is directly linked to an aspect of the respective object and can thus be easily controlled and manipulated. The compositional NeRF framework allows different, specialized NeRF architectures to be used for each object based on its type. To further improve quality, rather than directly rendering RGB images with the NeRF models, we render features and make use of a feature enhancer CNN to produce the RGB output. In order to represent objects with different appearances, we condition the NeRF and enhancer models on the style codes extracted with a dedicated style encoder [Menapace et al. 2022]. Our model is trained using reconstruction as the main guiding signal.

In Sec. 3.1.1-3.1.6 we illustrate the main components of the synthesis module and in Sec. 3.1.7 we describe the training procedure.

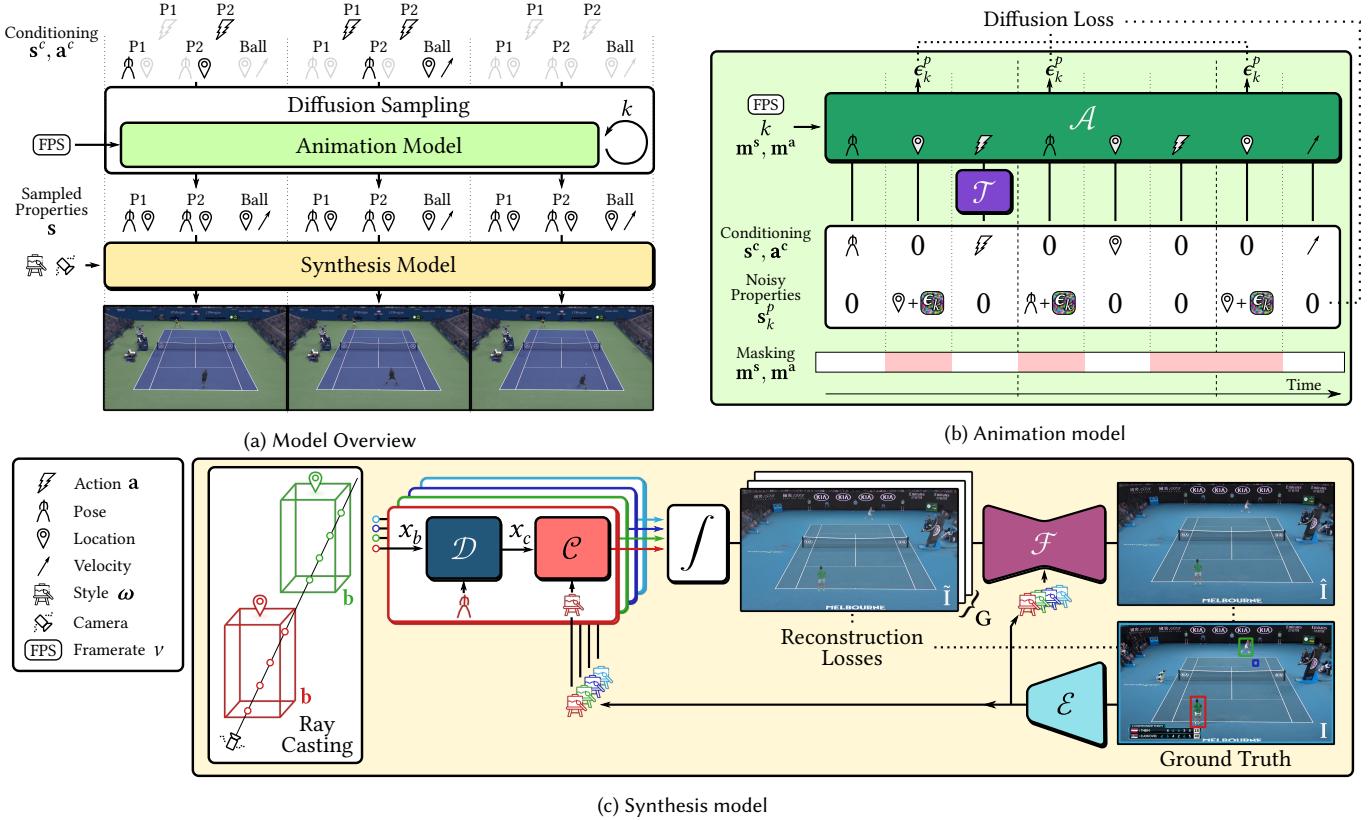


Fig. 2. (a) Overview of the LGE framework. The *animation model* produces states s based on user-provided conditioning signals s^c, a^c that are rendered by the *synthesis model*. (b) The diffusion-based animation model predicts noise ϵ_k applied to the noisy states s_k^p conditioned on known states s^c and actions a^c with the respective masks m^s, m^a , diffusion step k and framerate v . The text encoder \mathcal{T} produces embedding for the textual actions, while the temporal model \mathcal{A} performs noise prediction. (c) The synthesis model renders the current state using a composition of neural radiance fields, one for each object. A style encoder \mathcal{E} extracts the appearance ω of each object. Each object is represented in its canonical pose by C and deformations of articulated objects are modeled by the deformation model \mathcal{D} . After integration and composition, the feature grid G is rendered to the final image using the feature enhancer \mathcal{F} .

3.1.1 Scene Composition with NeRFs. Neural radiance fields represent a scene as a radiance field, a 5D function parametrized as a neural network mapping the current position x and viewing direction d to density σ and radiance c .

To allow controllable generation of complex scenes, we adopt a compositional strategy where each object in the scene is modeled with a dedicated NeRF model [Menapace et al. 2022; Müller et al. 2022; Xu et al. 2022a]. The scene is rendered by sampling points independently for each object and querying the respective object radiance field C_i .

All objects are assumed to be described by a set of properties whose structure depends on the type of object, e.g. a player, the ball, the background. We consider the following properties:

- **Object location.** Each object is contained within an axis-aligned bounding box b_i^{3D} which is defined by size and position. In the case of the ball, we additionally consider its velocity to model blur effects (Sec. 3.1.6).
- **Object style.** All objects have an appearance that may vary in different sequences, thus we introduce a style code ω_i as

an additional property for all objects. Since it is difficult to define such style information a priori, we assume it to be a latent variable and learn it jointly during training.

- **Object pose.** Articulated objects such as humans require additional properties to model varying poses. We model the deformation of articulated objects as a kinematic tree with J_i joints and consider as object properties the rotation R and translation tr parameters associated with each joint (Sec. 3.1.4).

From now on, we drop the object index i to simplify notation.

3.1.2 Style Encoder. Representing the appearance of each object is challenging since it changes based on the type of object and illumination conditions. We treat the style ω for each object as a latent variable that we regress using a convolutional style encoder \mathcal{E} . Given the current video frame I with O objects, we compute 2D bounding boxes b^{2D} for each object. First, a set of residual blocks is used to extract frame features which are later cropped around each object according to b^{2D} using ROI pooling [Girshick et al. 2013]. Later, a series of convolutional layers with a final projection is used to predict the style code ω from the cropped feature maps.

3.1.3 Volume Modeling for Efficient Sampling. Radiance fields are commonly parametrized using MLPs [Mildenhall et al. 2020] but such representation requires a separate MLP evaluation for each sampled point, making it computationally challenging to train high-resolution models. To overcome such issue, we model the radiance field C of each object in a canonical space using two alternative parametrizations.

For three-dimensional objects, we make use of a voxel grid parametrization [Fridovich-Keil et al. 2022; Weng et al. 2022]. Starting from a fixed noise tensor $V' \in \mathbb{R}^{F' \times H'_V \times W'_V \times D'_V}$, a series of 3D convolutions produces a voxel $V \in \mathbb{R}^{F+1 \times H_V \times W_V \times D_V}$ containing the features and density associated to each point in the bounded space. Here, F' and F represent the number of features, while H_V , W_V and D_V represent the size of the voxel. Given a point in the object canonical space x_c , the associated features and density σ are retrieved using trilinear sampling on V . To model the different appearance of each object, we adopt a small MLP conditioned on the style ω to produce a stylized feature with the help of weight demodulation [Karras et al. 2020].

For two-dimensional objects such as planar scene elements, we make use of a similar parametrization where a fixed 2D noise tensor $P' \in \mathbb{R}^{F' \times H'_P \times W'_P}$ is mapped to a plane of features $P \in \mathbb{R}^{F \times H_P \times W_P}$ using a series of 2D convolutions. Given a ray r , we compute the intersection point x between the plane and the ray which is used to sample P using bilinear sampling. Similarly to the voxel case, a small MLP is used to model object appearance according to ω . We assume planes to be fully opaque and assign a fixed density value σ to each sample. Thanks to this representation, a single point per ray is sufficient to render the object.

3.1.4 Deformation Modeling. Since the radiance field C alone supports only rendering of rigid objects expressed in a canonical space, to render articulated objects such as humans we introduce a deformation model \mathcal{D} . Given an articulated object, we assume its kinematic tree is known and that the transformation $[R_j | t_r]$ from each joint $j \in 1, \dots, J$ to the parent joint is part of the object's properties. We then implement a deformation procedure based on linear blend skinning (LBS) [Lewis et al. 2000] and inspired by Human-NeRF [Weng et al. 2022] that employs the joint transformations and a learned volume of blending weights $W \in \mathbb{R}^{J+1 \times H_W \times W_W \times D_W}$ to associate each point in the bounding box of the articulated object to the corresponding one in the canonical volume. We present additional details in Appx. C.

3.1.5 Enhancer. NeRF models are often parametrized to output radiance $c \in \mathbb{R}^3$ and directly produce an image. However, we find that such approach struggles to produce correct shading of the objects, with details such as shadows being difficult to synthesize. Also, to improve the computational efficiency of the method, we sample a limited number of points per ray that may introduce subtle artifacts in the geometry. To address these issues, we parametrize the model C to output features where the first three channels represent radiance and the subsequent represent learnable features. Then, we produce a feature grid $G \in \mathbb{R}^{F \times H \times W}$ and an RGB image $\hat{I} \in \mathbb{R}^{3 \times H \times W}$. We introduce an enhancer network \mathcal{F} modeled as a UNet [Ronneberger et al. 2015] architecture interleaved with weight

demodulation layers [Karras et al. 2020] that maps G and the style codes ω to the final RGB output $\hat{I} \in \mathbb{R}^{3 \times H \times W}$.

3.1.6 Object-specific rendering. Our compositional approach allows the use of object-specific techniques. In particular, in the case of tennis, we detail in Appx. B how we can apply dedicated procedures to enhance the rendering quality of the ball, the racket, and the 2D user interfaces such as the scoreboards. The rendering of the tennis ball is treated specially to render the blur that occurs in real videos in the case of fast-moving objects. The racket can be inserted in a post-processing stage to compensate for the difficulty of NeRFs to render thin, fast-moving objects. Finally, the UI elements are removed from the scene since they do not behave in a 3D consistent manner. For Minecraft, we describe how the scene skybox is modeled.

3.1.7 Training. We train our model using reconstruction as the main driving signal. Given a frame I and reconstructed frame \hat{I} , we use a combination of L2 reconstruction loss and the perceptual loss of Johnson et al. [Johnson et al. 2016] as our training loss. To minimize the alterations introduced by the enhancer and improve view consistency, we impose the same losses between I and \hat{I} , the output of the model before the feature enhancer. All losses are summed without weighting to produce the final loss term. To minimize GPU memory consumption, instead of rendering full images, we impose the losses on sampled image patches instead [Menapace et al. 2022].

We train all the components of the synthesis model jointly using Adam [Kingma and Ba 2015] for 300k steps with batch size 32. We set the learning rate to $1e-4$ and exponentially decrease it to $1e-5$ at the end of training. The framework is trained on videos with 1024x576px resolution. We present additional details in Appx. D.1 and in Appx. E.1.

3.2 Animation Model

In this section, we describe the animation model (see Fig. 2b), whose task is that of generating sequences of states $s \in \mathbb{S}^T$ according to user inputs. The animation model allows users to specify conditioning signals in two forms. First, conditional signals can take the form of values that the user wants to impose on some object properties in the sequence, such as the player position at a certain time step. This signal is represented by a sequence $s^c \in \mathbb{S}^T$. This form of conditioning allows fine control over the sequence to generate but requires directly specifying values of properties. Second, to allow high-level, yet granular control over the sequence, we introduce actions in the form of text $a^c \in \mathbb{L}^{A \times T}$ that specify the behavior of each of the A actionable objects at each timestep in the sequence, where \mathbb{L} is the set of all strings of text. To maximize the flexibility of the framework, we consider all values in s^c and a^c to be optional, thus we introduce their respective masks $m^s \in \{0, 1\}^{P \times T}$ and $m^a \in \{0, 1\}^{A \times T}$ that are set to 1 when the respective conditioning signal is present. We assume elements where the mask is not set to be equal to 0. The animation model predicts $s^p \in \mathbb{S}^T$ conditioned on s^c and a^c such that:

$$s = s^p + s^c, \quad (1)$$

where we consider the entries in s^p and s^c to be mutually exclusive, i.e. an element of s^p is 0 if the corresponding conditioning signal in

s^c is present according to m^s . Note that the prediction of actions is not necessary, since s is sufficient for rendering.

We adopt a temporal model \mathcal{A} based on a non-autoregressive masked transformer design and leverage the knowledge of a pre-trained language model in a text encoder \mathcal{T} to model action conditioning information [Han et al. 2022]. The masked design provides support for the optional conditioning signals and is trained using masked sequence modeling, where we sample m^s and m^a according to various strategies that emulate desired inference tasks.

In Sec. 3.2.1 we define our text encoder, Sec. 3.2.2 defines the diffusion backbone, and in Sec. 3.2.3 we describe the training procedure.

3.2.1 Text Encoder. We introduce a text encoder \mathcal{T} that encodes textual actions into a sequence of fixed-size text embeddings:

$$a^{\text{emb}} = \mathcal{T}(a^c) \in \mathbb{R}^{A \times T \times N_t}, \quad (2)$$

where N_t is the size of the embedding for the individual sentence. Given a textual action, we leverage a pretrained T5 text model [Raffel et al. 2022] \mathcal{T}_{enc} that tokenizes the sequence and produces an output feature for each token. Successively, a feature aggregator \mathcal{T}_{agg} modeled as a transformer encoder [Vaswani et al. 2017] produces the aggregated text embedding from the text model features. To retain existing knowledge into \mathcal{T}_{enc} , we keep it frozen and only train the feature aggregator \mathcal{T}_{agg} .

3.2.2 Temporal Modeling. In this section, we introduce the temporal model \mathcal{A} that predicts the sequence of states s conditioned on known state values s^c , action embeddings a^{emb} , and the respective masks m^s and m^a . Since only unknown state values need to be predicted, the model predicts s^p and the complete sequence of states is obtained as $s = s^p + s^c$, following Eq. (1). Diffusion models have recently shown state-of-the-art performance on several tasks closely related to our setting such as sequence modeling [Tashiro et al. 2021] and text-conditioned human motion generation [Dabral et al. 2022; Zhang et al. 2022]. Thus, we follow the DDPM [Ho et al. 2020] diffusion framework, and we frame the prediction of $s^p = s_0^p$ as a progressive denoising process s_0^p, \dots, s_K^p , where we introduce the diffusion timestep index $k \in 0, \dots, K$. The temporal model \mathcal{A} acts as a noise estimator that predicts the Gaussian noise ϵ_k in the noisy sequence of unknown states s_k^p at diffusion timestep k :

$$\epsilon_k^p = \mathcal{A}(s_k^p | s^c, a^{\text{emb}}, m^s, m^a, k). \quad (3)$$

An illustration of the proposed diffusion model is shown in Fig. 2b.

We realize \mathcal{A} using a transformer encoder [Vaswani et al. 2017]. To prepare the transformer’s input sequence, we employ linear projection layers \mathcal{P} with separate parameters for each object property. Since corresponding entries in s_k^p and s^c are mutually exclusive, we only consider the one that is present as input to the transformer and we employ different projection parameters to enable the model to easily distinguish between the two. An analogous projection is performed for a^{emb} and, subsequently, the projection outputs for states and actions are concatenated into a single sequence $e \in \mathbb{R}^{P+A \times T \times E}$, which constitutes the input to the transformer. An output projection layer with separate weights for each object property produces the prediction ϵ_k^p at the original dimensionality. To condition the model on the diffusion time-step k , we introduce a weight demodulation

layer [Karras et al. 2020] after each self-attention and feedforward block [Zhang et al. 2022].

To model long sequences while keeping reasonable computational complexity and preserving the ability to model long-term relationships between sequence elements, it is desirable to build the sequences using states sampled at a low framerate. However, this strategy would not allow the model to generate content at the original framerate and would prevent it from understanding dynamics such as limb movements that are clear only when observing sequences sampled at high framerates. To address this issue, we use the weight demodulation layers to further condition our model on the sampling framerate v to enable a progressive increase of the framerate at inference time (see Appx. F.2.1).

3.2.3 Training. To train our model, we sample a sequence s with corresponding actions a from a video in the dataset at a uniformly sampled framerate v . Successively, we obtain masks m^s and m^a according to masking strategies we detail in Appx. E.2. The sequence for training are obtained following $s_0^p = s \odot (1 - m^s)$ and $s^c = s \odot m^s$, and actions as $a^c = a \odot m^a$, where \odot denotes the Hadamard product.

We train our model by minimizing the DDPM [Ho et al. 2020] training objective:

$$\mathbb{E}_{k \sim \mathcal{U}(1, K), \epsilon \sim \mathcal{N}(0, I)} ||\epsilon_k^p - \epsilon_k||, \quad (4)$$

where ϵ_k^p is the noise estimated by the temporal model \mathcal{A} according to Eq. (3). Note that the loss is not applied to positions in the sequence corresponding to conditioning signals [Tashiro et al. 2021].

Our model is trained using the Adam [Kingma and Ba 2015] optimizer with a learning rate of $1e-4$, cosine schedule, and with 10k warmup steps. We train the model for a total of 2.5M steps and a batch size of 32. We set the length of the training sequences to $T = 16$. The number of diffusion timesteps is set to $K = 1000$ and we adopt a linear noise schedule [Ho et al. 2020]. Additional details are presented in Appx. D.2.

4 APPLICATIONS

Our Learnable Game Engine enables a series of applications that are unlocked by its expressive state representation, the possibility to render it using a 3D-aware synthesis model, and the ability to generate sequences of states with an animation model that understands the game logic and can be conditioned on a wide range of signals. In the following, we demonstrate a set of selected applications.

Our state representation is modular, where the style is one of the components. Style swapping is enabled by swapping the style of the desired object ω in the original image with the one from a target image. Similarly to a traditional game engine, our synthesis model renders the current state of the environment from a user-defined perspective. This enables our LGE to perform novel view synthesis. We show in Appx. G examples of both these capabilities.

We now show a set of applications enabled by the animation model. In Fig. 3, we show results for generating different sequences using textual actions starting from a common initial state. Thanks to the textual action representation, it is possible to gain fine control over the generated results and to make use of referential language.

Our animation model, however, is not limited to generating sequences given step-by-step actions. Thanks to its understanding

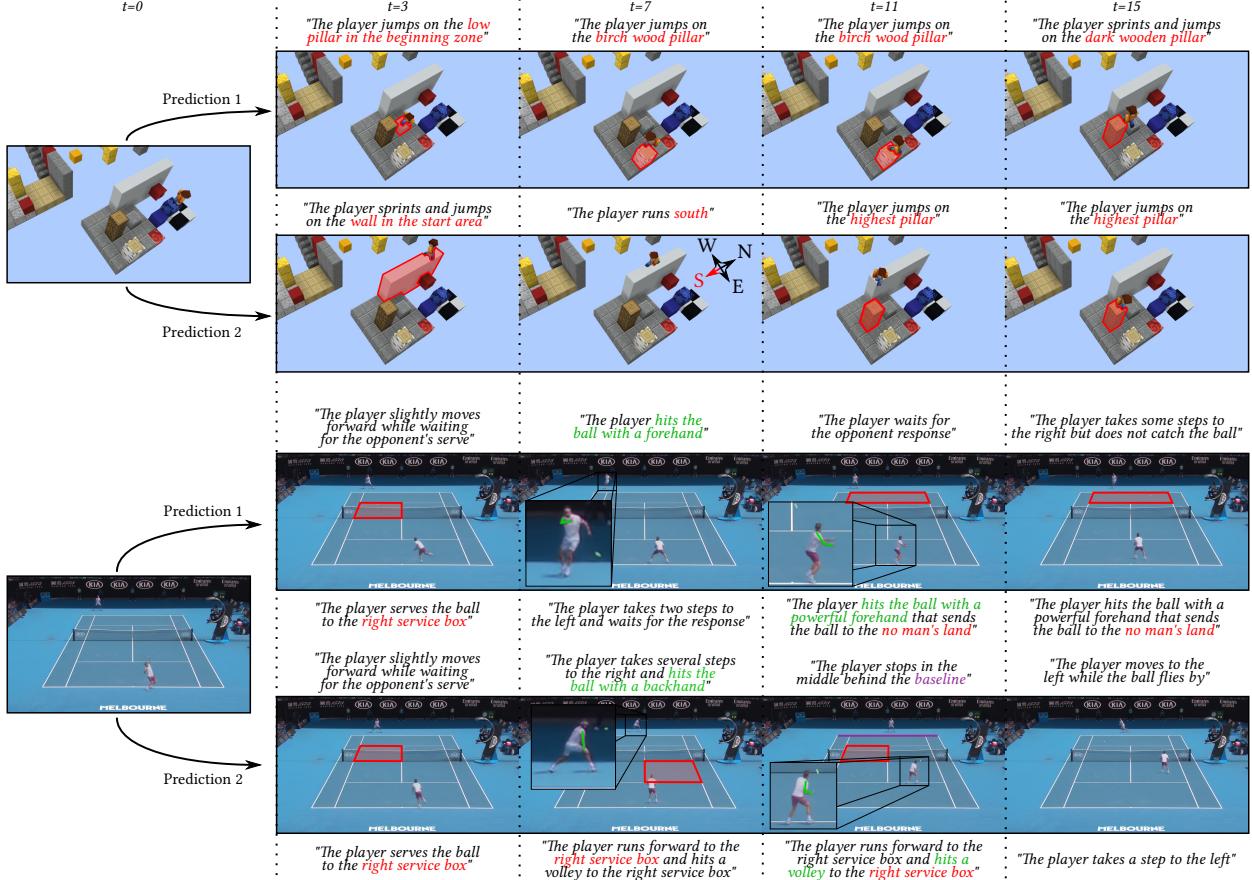


Fig. 3. Different sequences predicted on the Tennis and Minecraft datasets starting from the same initial state and altering the text conditioning. Our model moves players and designates shot targets using domain-specific referential language (eg. "right service box", "no man's land", "baseline"). The model supports fine-grained control over the various tennis shots using technical terms (eg. "forehand", "backhand", "volley"). See the Supp. Website for additional results.

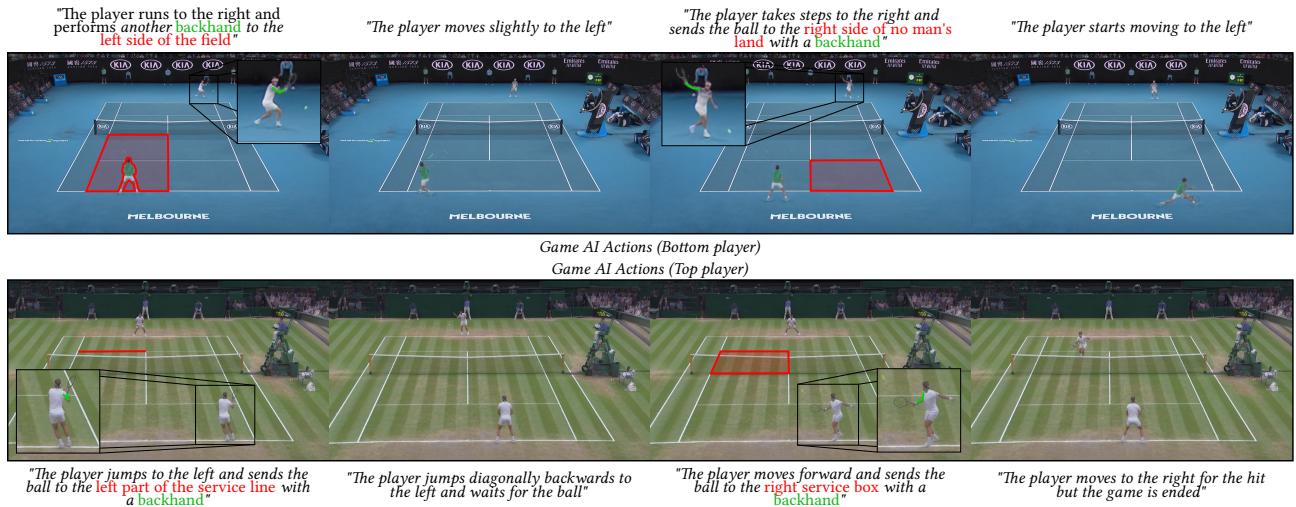


Fig. 4. Sequences generated by specifying actions for one of the players and letting the model act as the game AI and take control of the opponent. The game AI successfully responds to the actions of the player by running to the right (see top sequence) or towards the net (see bottom sequence), following two challenging shots of the user-controlled player.

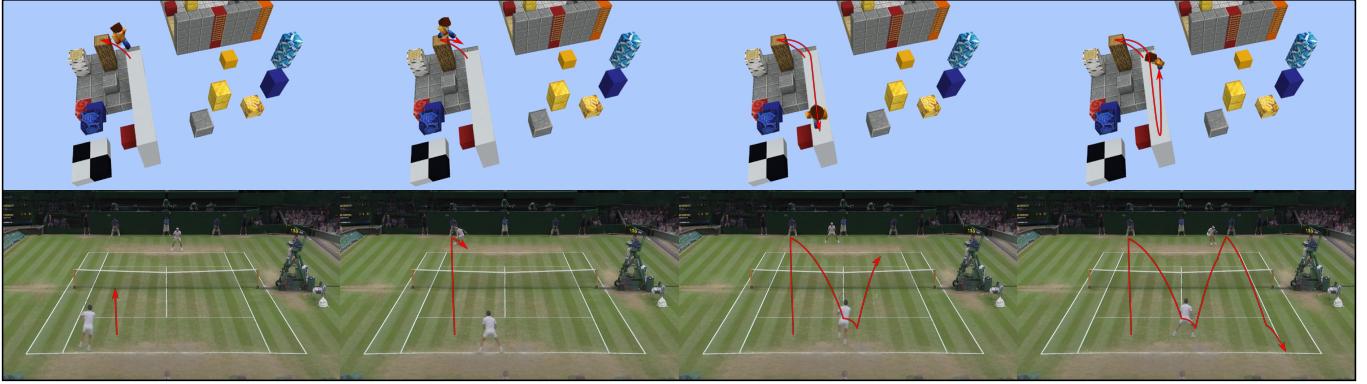


Fig. 5. Sequences generated without any user conditioning signal. The actions of all players are controlled by the model that acts as the game AI. In tennis, the players produce a realistic exchange, with the bottom player advancing aggressively toward the net and the top player defeating him with a shot along the right sideline. The Minecraft player and tennis ball trajectories are highlighted for better visualization. Please refer to the *Supp. Website* for additional results.

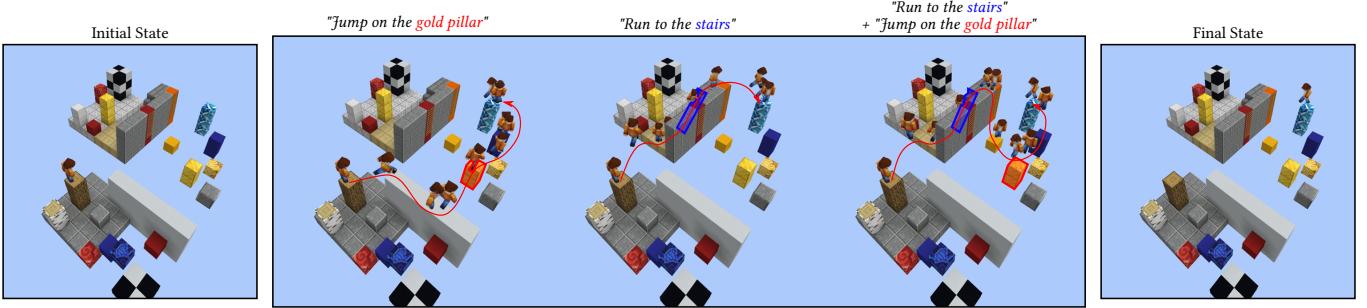


Fig. 6. Given an initial and a final state, we generate all the states in between. We repeat the generation multiple times conditioning it using different actions indicating the desired intermediate waypoints. Additional results are shown on the *Supp. Website*.

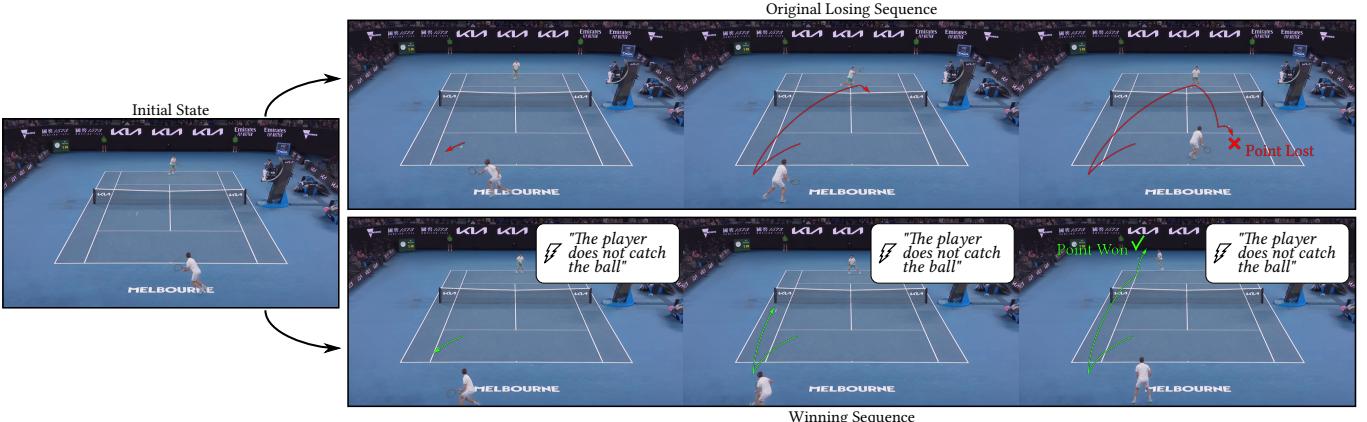


Fig. 7. Given a sequence where the bottom player loses (see top), we ask the model to modify it such that the bottom player wins instead (see bottom). To do so, we condition the top player on the action "*The player does not catch the ball*". While in the original sequence the bottom player aims its response to the center of the field where the opponent is waiting, the model now successfully generates a winning set of moves for the bottom player that sends the ball along the left sideline, too far for the top player to be reached. Video results are shown on the *Supp. Website*.

of the game’s logic, the model can tackle more complex tasks such as modeling an opponent against which a user-controlled player can play against (see Fig. 4), or even controlling all players in a scene without user intervention (see Fig. 5), in a way similar to a traditional game engine’s “game AI”.

The animation model also unlocks the “director’s mode”, where the user can generate sequences by specifying a desired set of high-level constraints or goals. The model is able to reason on actions to find a solution satisfying the given constraints. As a first example, Fig. 6 demonstrates results for a navigation problem, where the user specifies a desired initial and final player position in the scene, and the model devises a path between them. Notably, the user can also constrain the solution on intermediate waypoints by means of natural language. As a second example, Fig. 7 shows that the model is capable of devising strategies to defeat an opponent. Given an original sequence where the player commits a mistake and loses, the model can devise which actions the player should have taken to win. Notably, these model capabilities are learned by just observing sequences annotated with textual actions.

5 EVALUATION

In this section, we introduce our Tennis and Minecraft datasets (Sec. 5.1), describe our experimental protocol (Sec. 5.2), and perform evaluation of both the synthesis model (Sec. 5.3) and the animation model (Sec. 5.4).

5.1 Datasets

We collect two datasets to evaluate our method. Both datasets and the employed data collection tools are publicly available. In the following, we describe their structure and the available annotations.

5.1.1 Tennis dataset. We collect a dataset of broadcast tennis matches starting from the videos in [Menapace et al. 2022]. The dataset depicts matches between two professional players from major tennis tournaments, captured with a single, static bird’s eye camera.

To enable the construction of LGEs, we collect a wide range of annotations with a combination of manual and automatic methods (see Appx. A.1):

- For each frame, we perform camera calibration.
- For each of the two players, we perform tracking and collect full SMPL [Loper et al. 2015] body parameters. Note that in our work we only use a subset of the parameters: rotation and translation associated with each joint, and the location of the root joint in the scene.
- For each player and frame, we manually annotate textual descriptions of the action being performed. We structure captions so that each includes information on where and how the player is moving, the particular type of tennis shot being performed, and the location where the shot is aimed (see Appx. A.4). Captions make use of technical terms to describe shot types and field locations. In contrast to other video-text datasets that contain a single video-level [Bain et al. 2021] or high-level action descriptions weakly aligned with video content [Miech et al. 2019], the captions in our dataset are separate for each object and constitute a fine-grained description of the actions taking place in the frame.

- For the ball, we perform 3D tracking and provide its position in the scene and its velocity vector indicating the speed and direction of movement.

We collect 7112 video sequences in 1920x1080px resolution and 25fps starting from the videos in [Menapace et al. 2022] for a total duration of 15.5h. The dataset features 1.12M fully-annotated frames and 25.5k unique captions with 915 unique words. We highlight key statistics of the dataset show samples in Appx. A.

We note that broadcast Tennis videos are monocular and do not feature camera movements other than rotation, thus the dataset does not make it possible to recover the 3D geometry of static objects [Menapace et al. 2022].

5.1.2 Minecraft dataset. We collect a synthetic dataset from the Minecraft video game. This dataset depicts a player performing a series of complex movements in a static Minecraft world that include walking, sprinting, jumping, and climbing on various world structures such as platforms, pillars, stairs, and ladders. A single, monocular camera that slowly orbits around the scene center is used to capture the scenes. We collect a range of synthetic annotations using a game add-on we develop starting from [ReplayMod 2022]:

- Camera calibration for each frame.
- Player rotation and translation parameters associated with each joint in the Minecraft kinematic tree format, and the location of the root joint in the scene (see Appx. A.2).
- A synthetically-generated text caption describing the action being performed by the player. We assign varied, descriptive names to each element of the scene and build captions that describe scene elements or directions towards which the player is moving. Additionally, our captions capture how movement is happening i.e. by jumping, sprinting, walking, climbing, or falling. We adopt a stochastic caption generation procedure that generates multiple alternative captions for each frame.

A total of 61 videos are collected in 1024x576px resolution and 20fps for a total duration of 1.21h. The dataset contains 68.5k fully annotated frames and 1.24k unique captions with 117 unique words. We highlight key statistics for the dataset in Appx. A.

5.2 Evaluation Protocol

We evaluate the synthesis and the animation models separately, following a similar evaluation protocol. We divide the test dataset into non-overlapping sequences of 16 frames sampled at 5fps and 4fps respectively for the Minecraft and Tennis datasets and make use of the synthesis or animation model to reconstruct them. In the case of the synthesis model, we directly reconstruct the video frames and compute the following metrics:

- *LPIPS* [Zhang et al. 2018] is a standard metric for evaluating the reconstruction quality of the generated images
- *FID* [Heusel et al. 2017] is a widely-used metric for image generation quality
- *FVD* [Unterthiner et al. 2018] is a standard metric for assessing the quality of generated videos
- *Average Detection Distance (ADD)* [Menapace et al. 2021] measures the average distance in pixels between the bounding box centers of ground truth bounding boxes and bounding boxes



Fig. 8. Synthesis model qualitative results on the Tennis dataset. Compared to PE [Menapace et al. 2022], our model generates sharper players and static scene elements. Our ablation study shows corruption of the player geometry when voxels or our deformation model are not used. When removing our canonical plane representation, static scene elements appear blurry. When our feature enhancer is removed, the model does not generate shadows and players lose quality. We show video samples on the *Supp. Website*.

obtained from the generated sequences through a pretrained detector [Ren et al. 2015]

- *Missing Detection Rate (MDR)* [Menapace et al. 2021] estimates the rate of bounding boxes that are present in the ground truth, but that are missing in the generated videos

For the animation model, we evaluate reconstruction of the object properties. Note that different strategies for masking affect the behavior of the model and the nature of the reconstruction task, thus we separately evaluate different masking configurations corresponding to different inference tasks. We compute metrics that address both the fidelity of the reconstruction and the realism of the produced sequences:

- *L2* computes the fidelity of the reconstruction by measuring the distance between the ground truth and reconstructed object properties along the sequence
- *Fréchet Distance (FD)* [Fréchet 1957] measures the realism of each object property by computing the Fréchet Distance between the distribution of real sequences of a certain object property and of generated ones.

We select different reconstruction tasks for evaluation:

- *Video prediction conditioned on actions* consists in reconstructing the complete sequence starting from the initial state while the actions are specified for all timesteps. This setting corresponds to the evaluation setting of [Menapace et al. 2022].

- *Unconditioned video prediction* consists in reconstructing the complete sequence starting from the first state only.
- *Opponent modeling* consists in reconstructing the object properties of an unknown player, based on the state of the other player, with actions specified only on the known player. Good performance in this task indicates the ability to model an opponent against which a user can play.
- *Sequence completion* consists in reconstructing a sequence where 8 consecutive states are missing. No actions are specified for the missing states. Good performance in this task indicates ability in reasoning on how it is possible to reach a certain goal state starting from the current one.

5.3 Synthesis Model Evaluation

In this section, we evaluate the performance of the synthesis model.

5.3.1 Comparison to Baselines. We evaluate our method against Playable Environments (PE) [Menapace et al. 2022], the work most related to ours in that it builds a controllable 3D environment representation that is rendered with a compositional NeRF model where the position of each object is given and pose parameters are treated as a latent variable. Since the original method supports only outputs at 512x288px resolution, we produce baselines trained at both 512x288px and 1024x576px resolution which we name PE and PE+ respectively. For a fair comparison, we also introduce in the baselines

Table 1. Comparison with baselines and ablation of the synthesis model. MDR in %, ADD in pixels. Note that FID and FVD are computed on images downsampled to the feature extractor training resolution, thus blurriness in the PE baseline caused by its reduced resolution is not captured by these metrics. LPIPS correctly reflects lack of sharpness in the PE results (see Fig. 8). † denotes output in 512x288px rather than 1024x576px resolution.

Tennis	LPIPS↓	FID↓	FVD↓	ADD↓	MDR↓
PE† [Menapace et al. 2022]	0.188	11.5	349	3.74	0.200
PE+ [Menapace et al. 2022]	0.232	40.4	2432	132.3	49.7
w/o enhancer \mathcal{F}	0.167	15.6	570	3.02	0.0728
w/o explicit deformation in \mathcal{D}	0.156	13.3	524	3.10	0.0587
w/o planes in C	0.241	30.4	1064	2.94	0.0611
w/o voxels in C	0.170	17.1	757	3.03	0.0399
w/o our encoder \mathcal{E}	0.174	15.0	600	3.18	0.0564
Ours Small	0.156	13.4	523	2.88	0.0470
Ours	0.152	12.8	516	2.88	0.0423
Minecraft	LPIPS↓	FID↓	FVD↓	ADD↓	MDR↓
PE† [Menapace et al. 2022]	0.0235	13.9	21.5	5.77	0.0412
PE+ [Menapace et al. 2022]	0.0238	15.5	51.7	120.6	0.939
Ours Small	0.00996	3.56	8.83	2.02	0.0529
Ours	0.00814	2.81	7.08	1.98	0.0508

our same mechanism for representing ball blur and train a variant of our model using the same amount of computational resources as the baselines (Ours Small).

Results of the comparison are shown in Tab. 1, while qualitative results are shown in Fig. 8. Our method scores best in terms of LPIPS, ADD and MDR. Compared to PE+, our method produces significantly better FID and FVD scores. As shown in Fig. 8, PE and PE+ produce checkerboard artifacts that are particularly noticeable on static scene elements such as judge stands, while our method produces sharp details. We attribute this difference to our ray sampling scheme and feature enhancer design that, in contrast to PE, do not sample rays at low resolution and perform upsampling, but rather directly operate on high resolution. In addition, thanks to our deformation and canonical space modeling strategies, and higher resolution, our method produces more detailed players with respect to PE, where they frequently appear with missing limbs and blurred clothing. Finally, our model produces a realistic ball, while PE struggles to correctly model small objects, presumably due to its upsampling strategy that causes rays to be sampled more sparsely and thus do not intersect with the ball frequently enough to correctly render its blur effect. We show video results for both the Tennis and Minecraft datasets on the *Supp. Website*.

5.3.2 Ablation. To validate our design choices, we produce several variations of our method, each produced by removing one of our proposed architectural elements: we remove the enhancer \mathcal{F} and directly consider $\tilde{\mathcal{I}}$ as our output; we remove the explicit deformation modeling procedure in \mathcal{D} of Sec. 3.1.4 and substitute it with an MLP directly predicting the deformation using a learnable pose code as in [Menapace et al. 2022; Tretschk et al. 2021]; we remove the plane-based canonical volume representation in C for planar objects and use an MLP instead; we remove the voxel-based volume representation in C and use an MLP instead; we substitute our style

encoder \mathcal{E} with an ad-hoc encoder for each object in the scene, following [Menapace et al. 2022].

We perform the ablation on the Tennis dataset and show results in Tab. 1 and Fig. 8. To reduce computation, we train the ablation models using the same hyperparameters as the “Ours Small” model.

When removing the enhancer \mathcal{F} , our model produces players with fewer details and does not generate shadow effects below players (see first row in Fig. 8). When our deformation modeling procedure is not employed, the method produces comparable LPIPS, FID, and FVD scores, but an analysis of the qualitatives shows that players may appear with corrupted limbs (see last row in Fig. 8). In addition, the use of such learned pose representation would reduce the controllability of the synthesis model with respect to the use of an explicit kinematic tree. When plane-based or voxel-based canonical modeling is removed, we notice artifacts in the static scene elements, such as corrupted logos, and in the players, such as detached or doubled limbs. Finally, when we replace our style encoder design with the one of [Menapace et al. 2022], we notice fewer details in scene elements.

5.4 Animation Model Evaluation

In this section, we evaluate the performance of the animation model.

5.4.1 Comparison to Baselines. Similarly to the synthesis model, we compare our animation model against the one of Playable Environments (PE) [Menapace et al. 2022], the most related to our work since it operates on a similar environment representation. While the baseline jointly learns discrete actions and generates sequences conditioned on such actions, we assume the text action representations to be available in our task, so, for fairness of evaluation, we introduce our same text encoder \mathcal{T} in the baseline to make use of the action information. To reduce computation, we perform the comparison using half of the computational resources and a reduced training schedule, consequently, we also retrain our model, producing a reduced variant (Ours Small). To render results we always make use of our synthesis model.

We show results averaged over all inference tasks in Tab. 2 and report the results for each task in Appx. H.2. Our method outperforms the baseline in all evaluation tasks according to both L2 and FD metrics. From the qualitative results in Fig. 9 and in accordance with the FD metrics, we notice that our method produces more realistic player poses with respect to PE that tends to keep player poses close to the average pose and to slide the players on the scene. We attribute this difference to the use of the diffusion framework in our method. Consider the example of generating a player walking forward. It is equally probable that the player moves the left or right leg first. In the case of a reconstruction-based training objective such as the main one of PE, the model is encouraged to produce an average leg movement result that consists in not moving the legs at all. On the other hand, diffusion models learn the multimodal distributions of the motion, thus they are able to sample one of the possible motions without averaging its predictions.

5.4.2 Ablation. To validate this hypothesis and demonstrate the benefits of our diffusion formulation, we produce two variations of our method. The first substitutes the diffusion framework with a

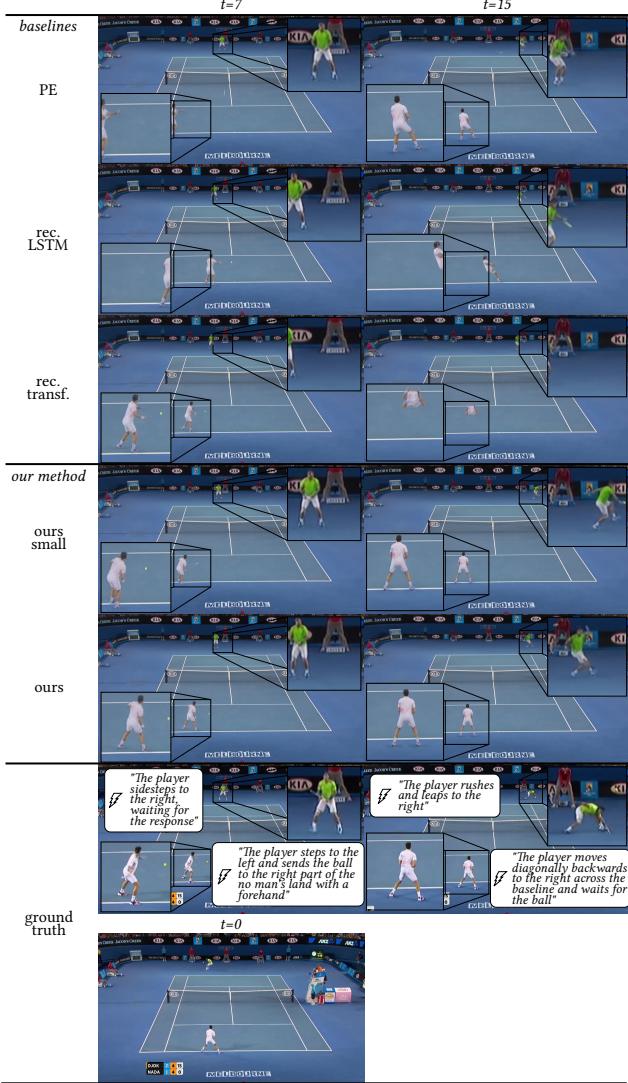


Fig. 9. Qualitative results on the Tennis dataset. Sequences are produced in a video prediction setting that uses the first frame object properties and all actions as conditioning. The location of players is consistently closer to the ground truth for our method. Our method captures the multimodal distribution of player poses and generates vivid limb movements, while the baselines produce poses as the average of the distribution, resulting in reduced limb movement and tilted root joints. Additional samples are shown in Appx. H.2 and in the *Supp. Website*.

reconstruction objective, keeping the transformer-based architecture unaltered. The second in addition to using the reconstruction objective models \mathcal{A} using an LSTM, similarly to the PE baseline. Differently from the PE baseline, however, this variant does not make use of adversarial training and employs a single LSTM model for all objects, rather than a separate model for each.

We show results in Tab. 2. Our model consistently outperforms the baselines in terms of FD, showing a better ability to capture realistic sequences. Consistently with our assessment in Sec. 5.4.1,

Table 2. Animation model comparison with baselines and ablation with results averaged over all inference tasks. Position and Joints 3D in meters, Root angle in axis-angle representation.

Tennis	Position		Root angle		Joints 3D	
	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓
PE	3.291	229.112	1.126	15.953	0.303	53.242
Rec. LSTM	1.597	7.253	0.907	7.051	0.193	16.735
Rec. Transf.	1.074	4.402	0.767	6.838	0.175	14.845
Ours Small	1.380	1.443	1.014	0.560	0.148	1.253
Ours	1.099	0.929	0.844	0.356	0.129	0.836
Minecraft	Position		Root angle		Joints 3D	
	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓
PE	2.739	105.973	1.620	31.232	0.311	39.572
Rec. LSTM	2.292	47.296	1.702	49.971	0.489	99.843
Rec. Transf.	2.154	53.198	1.430	36.123	0.385	69.977
Ours Small	1.084	4.461	1.077	6.016	0.140	3.590
Ours	1.065	4.815	0.956	4.083	0.132	3.360

Fig. 9 shows that our method trained with a reconstruction objective produces player movement with noticeable artifacts analogously to PE, validating the choice of the diffusion framework.

6 CONCLUSIONS

In this paper, we demonstrate the feasibility of learning game engines solely from annotated data and show that textual action representations are critical for unlocking high-level and fine-grained control over the generation process, and enabling compelling constraint- and goal-driven generation applications. These results, jointly with two richly-annotated text-video datasets, pave the way towards learning game engines for complex, real-world scenes.

7 ACKNOWLEDGEMENTS

We would like to thank Denys Poluyanov, Eugene Shevchuk and Oleksandr Pyshchenko for the useful discussion and validation of the use cases of the LGE framework, Maryna Diakonova for her support in data labeling, and Anton Kuzmenko and Vadym Hrebennik for their assistance in creating the accompanying video.

REFERENCES

- Panos Achlioptas, Ian Huang, Minhyuk Sung, Sergey Tulyakov, and Leonidas Guibas. 2023. ShapeTalk: A Language Dataset and Framework for 3D Shape Edits and Deformations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Nikos Athanasiou, Mathis Petrovich, Michael J. Black, and Gürkaynak Varol. 2022. TEACH: Temporal Action Compositions for 3D Humans. In *International Conference on 3D Vision (3DV)*.
- Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. 2018. Stochastic Variational Video Prediction. In *International Conference on Learning Representations (ICLR)*.
- Max Bain, Arsha Nagrani, Gürkaynak Varol, and Andrew Zisserman. 2021. Frozen in Time: A Joint Video and Image Encoder for End-to-End Retrieval. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
- Michael Büttner and Simon Clavet. 2015. Motion Matching - The Road to Next Gen Animation. In *Proc. of Ntu.ai 2015*.
- Zhe Cao, Hang Gao, Karttikeya Mangalam, Qizhi Cai, Minh Vo, and Jitendra Malik. 2020. Long-term human motion prediction with scene context. In *Proceedings of the European Conference of Computer Vision (ECCV)*.
- Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Samih Khamis, Tero Karras, and Gordon Tremblay. 2022. Efficient Geometry-aware 3D Generative Adversarial Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

- Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *European Conference on Computer Vision (ECCV)*.
- Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, and William Chan. 2021. WaveGrad: Estimating Gradients for Waveform Generation. In *International Conference on Learning Representations (ICLR)*.
- Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. 2017. Recurrent Environment Simulators. *CoRR* abs/1704.02254 (2017). arXiv:1704.02254
- Hongsuk Choi, Gyeongsik Moon, JoonKyu Park, and Kyoung Mu Lee. 2022. Learning to Estimate Robust 3D Human Mesh from In-the-Wild Crowded Scenes. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Robert L. Cook, Thomas Porter, and Loren Carpenter. 1984. Distributed Ray Tracing. *SIGGRAPH Comput. Graph.* 18, 3 (jan 1984), 137–145.
- Cassidy Curtis, Sigrurdur Orn Adalgeirsson, Horia Stefan Ciurdar, Peter McDermott, JD Velásquez, W. Bradley Knox, Alonso Martinez, Dei Gaztelumendi, Norberto Adrian Goussies, Tianyu Liu, and Palash Nandy. 2022. Toward Believable Acting for Autonomous Animated Characters. In *Proceedings of the 15th ACM SIGGRAPH Conference on Motion, Interaction and Games*.
- Rishabh Dabral, Muhammad Hamza Mughal, Vladislav Golyanik, and Christian Theobalt. 2022. MoFusion: A Framework for Denoising-Diffusion-based Motion Synthesis.
- Aram Davtyan and Paolo Favaro. 2022. Controllable Video Generation through Global and Local Motion Dynamics. In *Proceedings of the European Conference of Computer Vision (ECCV)*.
- Sander Dieleman, Laurent Sartran, Arman Roshnaii, Nikolay Savinov, Yaroslav Ganin, Pierre H. Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, Curtis Hawthorne, Rémi Leblond, Will Grathwohl, and Jonas Adler. 2022. Continuous diffusion for categorical data.
- Dirk Farin, Susanne Krabbe, Peter H. N. de With, and Wolfgang Effelsberg. 2003. Robust camera calibration for sport videos using court models. In *IS&T/SPIE Electronic Imaging*.
- Vincent Fortuin, Dmitry Baranchuk, Gunnar Rätsch, and Stephan Mandt. 2020. Gp-vae: Deep probabilistic time series imputation. In *International conference on artificial intelligence and statistics*. PMLR, 1651–1661.
- Maurice Fréchet. 1957. Sur la distance de deux lois de probabilité. *Comptes Rendus Hebdomadaires des Séances de L Académie des Sciences* 244, 6 (1957), 689–692.
- Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5491–5500.
- Tsu-Jui Fu, Licheng Yu, Ning Zhang, Cheng-Yang Fu, Jong-Chyi Su, William Yang Wang, and Sean Bell. 2022. Tell Me What Happened: Unifying Text-guided Video Completion via Multimodal Masked Video Generation.
- Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2013. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (11 2013).
- Ligong Han, Jian Ren, Hsin-Ying Lee, Francesca Barbieri, Kyle Olszewski, Sherwin Minnae, Dimitris Metaxas, and Sergey Tulyakov. 2022. Show Me What and Tell Me How: Video Synthesis via Multimodal Conditioning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.
- Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 30. 6626–6637.
- Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P. Kingma, Ben Poole, Mohammad Norouzi, David J. Fleet, and Tim Salimans. 2022a. Imagen Video: High Definition Video Generation with Diffusion Models.
- Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 6840–6851.
- Jonathan Ho, Tim Salimans, Alexey A. Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. 2022b. Video Diffusion Models. In *ICLR Workshop on Deep Generative Models for Highly Structured Data*.
- Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. 2018. CyCADA: Cycle-Consistent Adversarial Domain Adaptation. In *Proceedings of the 35th International Conference on Machine Learning (ICML) (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 1989–1998.
- Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. 2020. Learned Motion Matching. *ACM Transactions on Graphics (TOG)* 39, 4, Article 53 (aug 2020), 13 pages.
- Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)* 36 (2017), 1 – 13.
- Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, and Jie Tang. 2022. CogVideo: Large-scale Pretraining for Text-to-Video Generation via Transformers. (2022).
- Jiahui Huang, Yuhe Jin, Kwang Moo Yi, and Leonid Sigal. 2022. Layered Controllable Video Generation. In *Proceedings of the European Conference of Computer Vision (ECCV)*, Shai Aviad, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.).
- Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel, and Ben Poole. 2022. Zero-Shot Text-Guided Object Generation with Dream Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *Proceedings of the European Conference of Computer Vision (ECCV)*.
- Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. Analyzing and Improving the Image Quality of StyleGAN. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. 2021. DriveGAN: Towards a Controllable High-Quality Neural Simulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5820–5829.
- Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. 2020. Learning to Simulate Dynamic Environments with GameGAN. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).
- Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. 2020. DiffWave: A Versatile Diffusion Model for Audio Synthesis. In *International Conference on Learning Representations (ICLR)*.
- Zhenfei Kuang, Kyle Olszewski, Menglei Chai, Zeng Huang, Panos Achlioptas, and Sergey Tulyakov. 2022. NeROIC: Neural Rendering of Objects from Online Image Collections. *ACM Transactions on Graphics (TOG)* 41, 4, Article 56 (jul 2022), 12 pages.
- Abhijit Kundu, Kyle Genova, Xiaoli Yin, Alireza Fathi, Caroline Pantofaru, Leonidas J. Guibas, Andrea Tagliasacchi, Frank Dellaert, and Thomas A. Funkhouser. 2022. Panoptic Neural Fields: A Semantic Object-Aware Neural Scene Representation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), 12861–12871.
- Yong-Hoon Kwon and Min-Gyu Park. 2019. Predicting future frames using retrospective cycle gan. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1811–1820.
- Max W. Y. Lam, Jun Wang, Dan Su, and Dong Yu. 2022. BDDM: Bilateral Denoising Diffusion Models for Fast and High-Quality Speech Synthesis. In *International Conference on Learning Representations (ICLR)*.
- Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive Character Animation by Learning Multi-Objective Control. *ACM Transactions on Graphics (TOG)* 37, 6, Article 180 (dec 2018), 10 pages.
- Yichong Leng, Zehua Chen, Junliang Guo, Haohu Liu, Jiawei Chen, Xu Tan, Danilo Mandic, Lei He, Xiangyang Li, Tao Qin, sheng zhao, and Tie-Yan Liu. 2022. Binural-Grad: A Two-Stage Conditional Diffusion Probabilistic Model for Binural Audio Synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.).
- Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. 2009. EPnP: An Accurate O(n) Solution to the PnP Problem. (2009).
- J. P. Lewis, Matt Cordiner, and Nickson Fong. 2000. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation.
- Ruilong Li, Julian Tanke, Minh Vo, Michael Zollhofer, Jürgen Gall, Angjoo Kanazawa, and Christoph Lassner. 2022. TAVA: Template-free animatable volumetric actors. In *Proceedings of the European Conference of Computer Vision (ECCV)*.
- Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. 2022. Magic3D: High-Resolution Text-to-3D Content Creation.
- Lingjie Liu, Marc Habermann, Viktor Rudnev, Kripasindhu Sarkar, Jiatao Gu, and Christian Theobalt. 2021. Neural Actor: Neural Free-View Synthesis of Human Actors with Pose Control. *ACM Transactions on Graphics (TOG)* 40, 6, Article 219 (dec 2021), 16 pages.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Dangi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. RoBERTa: A Robustly Optimized {BERT} Pretraining Approach. In *International Conference on Learning Representations (ICLR)*.
- Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Transactions on Graphics (TOG)* 34, 6, Article 248 (nov 2015), 16 pages.
- Willi Menapace, Stephane Lathuilière, Sergey Tulyakov, Aliaksandr Siarohin, and Elisa Ricci. 2021. Playable Video Generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 10061–10070.
- Willi Menapace, Stéphane Lathuilière, Aliaksandr Siarohin, Christian Theobalt, Sergey Tulyakov, Vladislav Golyanik, and Elisa Ricci. 2022. Playable Environments: Video Manipulation in Space and Time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

- Chenlin Meng, Ruiqi Gao, Diederik P Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. 2022. On Distillation of Guided Diffusion Models. In *NeurIPS 2022 Workshop on Score-Based Methods*.
- Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. 2019. HowTo100M: Learning a Text-Video Embedding by Watching Hundred Million Narrated Video Clips. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
- 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 *Proceedings of the European Conference of Computer Vision (ECCV)*.
- Norman Müller, Andrea Simonelli, Lorenzo Porzi, Samuel Rota Bulò, Matthias Nießner, and Peter Kontschieder. 2022. AutoRF: Learning 3D Object Radiance Fields from Single View Observations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Transactions on Graphics (TOG)* 41, 4, Article 102 (July 2022), 15 pages.
- Michael Niemeyer and Andreas Geiger. 2021. GIRAFFE: Representing Scenes As Compositional Generative Neural Feature Fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 11453–11464.
- Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. 2015. Action-conditioned video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2863–2871.
- Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. 2021. Neural Scene Graphs for Dynamic Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2856–2865.
- Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. 2021a. Nerfies: Deformable Neural Radiance Fields. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2021).
- Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. 2021b. HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields. *ACM Transactions on Graphics (TOG)* 40, 6, Article 238 (dec 2021).
- Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. 2022. DreamFusion: Text-to-3D using 2D Diffusion.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *International Conference on Machine Learning (ICML)*.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2022. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21, 1, Article 140 (jun 2022), 67 pages.
- Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical Text-Conditional Image Generation with CLIP Latents. *ArXiv abs/2204.06125* (2022).
- Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-Shot Text-to-Image Generation. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 8821–8831.
- Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 28. 91–99.
- ReplayMod. 2022. ReplayMod. <https://github.com/ReplayMod/ReplayMod>. Accessed: 2021-11-12.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2021. High-Resolution Image Synthesis with Latent Diffusion Models. *arXiv:2112.10752 [cs.CV]*
- Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (Eds.). Springer International Publishing, Cham, 234–241.
- Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. 2022. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *ArXiv abs/2205.11487* (2022).
- Tim Salimans and Jonathan Ho. 2022. Progressive Distillation for Fast Sampling of Diffusion Models. In *International Conference on Learning Representations (ICLR)*.
- Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 464–468.
- Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. 2022. Make-A-Video: Text-to-Video Generation without Text-Video Data.
- Jiaming Song, Chenlin Meng, and Stefano Ermon. 2021. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations (ICLR)*.
- Matt Stanton, Sascha Gedder, Adrian Blumer, Paul Hormis, Andy Nealen, Seth Cooper, and Adrien Treuille. 2016. Large-Scale Finite State Game Engines. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*.
- Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. 2019. Neural State Machine for Character-Scene Interactions. *ACM Transactions on Graphics (TOG)* 38, 6 (2019).
- Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. 2020. Local Motion Phases for Learning Multi-Contact Character Movements. *ACM Transactions on Graphics (TOG)* 39, 4, Article 54 (aug 2020), 14 pages.
- Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. 2021. CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Guy Tevet, Brian Gordon, Amir Hertz, Amit H. Bermano, and Daniel Cohen-Or. 2022. MotionCLIP: Exposing Human Motion Generation to CLIP Space. In *Proceedings of the European Conference of Computer Vision (ECCV) (Lecture Notes in Computer Science, Vol. 13682)*, Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.). Springer, 358–374.
- Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. 2021. Non-Rigid Neural Radiance Fields: Reconstruction and Novel View Synthesis of a Dynamic Scene From Monocular Video. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
- Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. 2018. Towards Accurate Generative Models of Video: A New Metric & Challenges. *ArXiv preprint arXiv:1812.01717* (2018).
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc.
- Chung-Yi Weng, Brian Curless, Pratul P. Srinivasan, Jonathan T. Barron, and Ira Kemelmacher-Shlizerman. 2022. HumanNeRF: Free-Viewpoint Rendering of Moving People From Monocular Video. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 16210–16220.
- Tianhao Walter Wu, Fangcheng Zhong, Andrea Tagliasacchi, Forrester Cole, and Cengiz Oztireli. 2022. D^A2NeRF: Self-Supervised Decoupling of Dynamic and Static Objects from a Monocular Video. In *Advances in Neural Information Processing Systems (NeurIPS)*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.).
- Yinghao Xu, Menglei Chai, Zifan Shi, Sida Peng, Skorokhodov Ivan, Siarohin Aliaksandr, Ceyuan Yang, Yujun Shen, Hsin-Ying Lee, Bolei Zhou, and Tulyakov Sergy. 2022a. DiscoScene: Spatially Disentangled Generative Radiance Field for Controllable 3D-aware Scene Synthesis. *arxiv: 2212.11984* (2022).
- Yufei Xu, Jing Zhang, Qiming Zhang, and Dacheng Tao. 2022b. ViTPose: Simple Vision Transformer Baselines for Human Pose Estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
- Mingyu Zhang, Zhongang Cai, Liang Pan, Fangzhou Hong, Xinying Guo, Lei Yang, and Ziwei Liu. 2022. MotionDiffuse: Text-Driven Human Motion Generation with Diffusion Model.
- Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. 2018. ICNet for Real-Time Semantic Segmentation on High-Resolution Images. In *Proceedings of the European Conference of Computer Vision (ECCV)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 418–434.
- Yi Zhou, Connally Barnes, Lu Jingwan, Yang Jimei, and Li Hao. 2019. On the Continuity of Rotation Representations in Neural Networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

A DATASETS

In this section, we give additional details on the dataset, including the dataset collection process, Minecraft 3D skeleton format, additional dataset statistics, and dataset samples.

A.1 Tennis Dataset Collection

We build our tennis dataset starting from the one of [Menapace et al. 2022]. However, we notice that such dataset has an imprecise camera calibration and lacks information such as 3D player poses and 3D localization of the ball. Thus, we only retain the original videos and acquire new annotations. We describe the process in the following sections and release all code related to dataset creation.

A.1.1 Camera Calibration. To improve camera calibration, we notice that the original dataset bases its camera calibration on field keypoints detected using [Farin et al. 2003], but such keypoint estimates are noisy. To overcome this issue, we manually annotate a subset of 10569 frames with field keypoint information and train a keypoint detection model inspired by ICNet [Zhao et al. 2018], which we choose due to its reduced memory footprint which allows us to train the model in full 1920x1080px resolution for best results. The detected keypoints are filtered and used to produce camera calibration. Compared to the camera calibration of [Menapace et al. 2022], we notice less jitter and are able to successfully perform camera calibration on a larger number of video instances.

A.1.2 3D Ball Localization. To produce 3D ball localization, we first build a 2D ball detector following the same approach used for field keypoints localization, starting from 17330 manually annotated frames. In addition to 2D ball localization, we manually annotate the projection of the ball on the field plane for a set of keyframes defined as the frames where contact between the ball and an object different than the field happens or the first and last frames of the video with a visible ball. The field plane projections of the ball in conjunction with the camera calibration results and 2D ball detections can be used to recover the 3D ball position in those frames.

We assume that between the keyframes, no contact happens that significantly alters the horizontal speed of the ball apart from air drag. In practice, contact between the ball and the field during bounces does affect ball speed, and we take account of it in a second, refinement phase. We thus model the horizontal ball position on the line between the ball positions at two consecutive keyframes by solving the linear motion equation under air drag:

$$x(t) = x_0 \frac{\log(1 + Cv_0 t)}{C}, \quad (5)$$

where x_0 is the initial position, v_0 is the initial velocity, t is time and C is an estimated coefficient summarizing fluid viscosity, drag coefficient, and shape of the ball. Note that the effects of gravity are ignored in the equation. C can be estimated by inverting Eq. (5), based on initial ball speed measurements for v_0 that can be extracted from the videos thanks to the service ball speed radars installed on tennis fields, and the positions the ball at keyframes. Given the ball's horizontal position on the line joining the 3D ball position at the preceding and succeeding keyframes, we can recover its 3D position by intersecting the camera ray passing from the 2D projection of the ball on that frame with the plane parallel to the net that intersects with the ball's horizontal position.

To improve the precision of results and account for horizontal ball speed changes during bounces, in a second phase we detect bounces between the ball and the field and impose that the ball

touches the field at those positions, by considering them as additional keyframes and repeating the procedure. Finally, to calibrate frames with missing 2D ball detections (eg. ball thrown high above the camera frames or heavy blur and image compression artifacts), we recover the ball position by fitting a ballistic trajectory using 3D ball localization from neighboring frames.

A.1.3 3D Player Poses. To recover 3D player poses, we rely on the 3DCrowdNet pose estimator [Choi et al. 2022] which we find robust to the presence of frequent overlaps between players and referees, player limbs blur, and low player resolution. 3DCrowdNet assumes 2D joint locations to be given as input, so we produce them using the state-of-the-art 2D pose estimator VitPose [Xu et al. 2022b] which we find robust to blur, reduced player size, and occlusions. The extracted 3D skeletons however are expressed under the coordinate system of a framework-predicted camera. We make use of a PnP [Lepetit et al. 2009] procedure to register the 3D skeletons to our calibrated camera and reduce depth estimation errors by placing the estimated 3D skeletons with their feet touching the ground. Note that, while 3DCrowdNet regresses full SMPL [Loper et al. 2015] parameters and meshes, we only make use of 3D joint locations and joint angles. SMPL body shape parameters are nevertheless included in the dataset to support its different use cases.

A.1.4 Text Action Annotation. We manually annotate each video sequence using a text caption for each player and frame. Each caption focuses on the action being performed by the player in that instant and captures several aspects of the action. The caption captures where the player is moving and how the player is moving, i.e. the player is running, walking, sliding, or falling, the player is moving to its left, towards the net, across the baseline. When a player is performing a ball-hitting action, the particular type of tennis shot being performed is presented, e.g. a smash, a serve, a lob, a backhand, a volley, and the location where the ball is aimed is described. We report text annotation statistics in Tab. 3.

A.1.5 UI Elements Annotation. We manually annotate each video sequence with a set of 2D bounding boxes indicating the places where 2D UI elements such as scoreboards or tournament logos may appear during the sequence.

A.2 Minecraft Skeleton Format

We adopt a skeletal player representation that divides the Minecraft body into 6 parts: head, torso, left and right arm, and left and right leg. We place 6 corresponding joints at the bottom of the head, top of the torso, shoulders, and top of the legs. Following the internal Minecraft skeletal representation, a root joint is added that is the parent of the 6 joints. We extend this representation by introducing 6 additional joints at the top of the head, top of the torso, bottom of the arms, and bottom of the legs. The additional joints have as parents the original joint positioned on the same body part. While the additional 6 joints are always associated with a zero rotation, we find their introduction convenient for skeleton visualization purposes. Fig. 10 provides a visualization of such skeletons.

A.3 Additional Dataset Statistics

We provide the main dataset statistics in Tab. 3, with additional ones in Fig. 11, where we plot the distribution of video lengths in the dataset and the average number of words in each caption. The Tennis dataset features manually-annotated captions which contain a greater number of words with respect to the synthetic annotations in the Minecraft dataset.

A.4 Dataset Samples

We show samples from the Minecraft and Tennis dataset in Fig. 10 and show samples in the form of videos on the *Supp. Website*.

We now show a non-curated set of captions extracted from the Tennis dataset:

- “the player prepares to hit the ball but stops, opponent hits the net”
- “the player starts to move to the left when the opponent sends the ball out of the field”
- “the player moves diagonally to the right and forward to the right side of the baseline and sends the ball to the right side of no man’s land with a forehand”
- “the player takes sidestep to the right and hits the ball with a backhand that sends the ball to the right side of the no man’s land”
- “the player moves left to hit the ball but stops halfway”
- “the player sidesteps to the left and stops, because the ball goes out of bounds”

We report a set of peculiar words extracted from the set of words with the lowest frequency on the Tennis dataset: “scratching”, “inertia”, “previously”, “realize”, “understands”, “succeed”, “bind”, “touched”, “circling”, “approaching”, “bolting”, “entering”, “ducks”, “reaction”, “repeat”, “wipes”, “abruptly”, “preparation”, “dramatic”, “soft”, “celebrating”, “losing”, “strides”, “dart”, “reacts”, “block”, “sideway”, “ending”, “becomes”, “dismissively”, “continuous”, “squat”, “says”, “intends”, “ricochet”, “delays”, “night”, “guess”, “manage”, “already”, “correctly”, “anticipation”, “unsuccessfully”, “inaccurate”, “deflection”, “properly”, “swinging”.

We show a non-curated set of captions extracted from the Minecraft dataset:

- “the player falls on the birch pillar”
- “the player moves fast north, jumps”
- “the player jumps on the intermediate wooden pillar”
- “the player falls on the platform opposite to the stairs”
- “the player runs to the big stone platform”
- “the player climbs down and does not rotate”
- “the player moves south east, jumps and rotates counterclockwise”
- “the player runs to the red decorated block”

We list a set of peculiar words from the Minecraft dataset: “nothing”, “facing”, “space”, “level”, “map”, “leading”, “opposite”, “edge”.

B OBJECT-SPECIFIC SYNTHESIS MODEL TECHNIQUES

The compositional nature of the synthesis module makes it possible to adopt object-specific techniques to model particular objects. In the following, we describe the techniques adopted to model balls

Table 3. Dataset statistics for the Tennis and Minecraft datasets.

	Tennis	Minecraft
Sequences:	7112	61
train	5690	51
validation	711	5
test	711	5
Duration:	15.5h	1.21h
train	12.4h	0.952h
validation	1.59h	0.16h
test	1.52h	0.101
Annotated frames:	1.12M	68.5k
train	1.05M	64.5k
validation	135k	11.2k
test	130k	7.06k
Resolution	1920x1080px	1024x576px
Framerate	25fps	20fps
Captions	84.1k	818k
of which unique	25.5k	1.24k
Unique words	915	117
Avg. words	13.8	5.85
Avg. span	1.32s	0.500s
Parts of sentence:		
Nouns	32.3%	36.2%
Verbs	11.9%	17.4%
Adjectives	3.08%	6.48%
Adverbs	2.70%	11.7%
Pronouns	0.18%	0.00%
Articles	26.4%	8.03%
Prepositions	7.89%	6.98%
Numerals	0.11%	0.03%
Particles	9.28%	1.50%
Punctuation	1.76%	1.12%
Others	0.00%	0.00%

(Appx. B.1), rackets (Appx. B.2), 2D UI elements (Appx. B.3), and skyboxes (Appx. B.4).

B.1 Ball Modeling

Fast-moving objects may appear blurred in real video sequences. This effect is frequent in ball objects found in sports videos and is thus desirable to model this effect. To model them, we adopt a procedure inspired by [Cook et al. 1984], which distributes multiple rays in time to model blur effects. We extend the object properties of the ball object to also include a velocity vector v . Given the ball radius r and an estimate for the shutter speed t_c of the camera, we can compute in closed form the probability p that a given point in space intersects with the ball object while the ball moves during the time the camera shutter remains open to capture the current frame. To model blur, we assign to each point a fixed density multiplied by p . Modeling p in closed form avoids the need to sample multiple rays in time, improving performance.

To compute p (see Fig. 12), we first use the velocity vector v to estimate the rotation R_b that maps each point x_b in the ball bounding box to a canonical space x_c in which the ball velocity vector is aligned to the positive y -axis $x_c = R_b x_b$. Then we compute the distance traveled by the ball while the shutter remains open $d = \|v\|_2 t_c$. We then compute the useful cross-section of the ball d_y that can intersect with x_c as the diameter of the circumference originating from the intersection between the ball and a plane with a distance from the ball center r_y equal to the distance of x_c from



Fig. 10. Sampled frames from the Tennis (left) and Minecraft (right) datasets. 3D skeletons are annotated in blue, while the 3D ball is visualized in green.

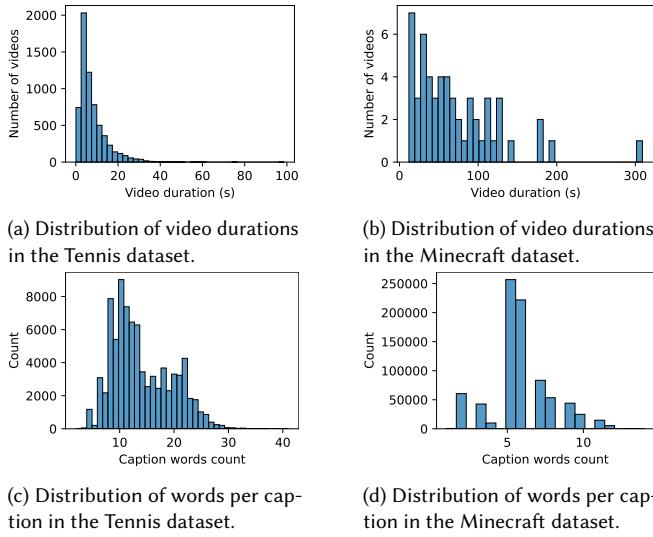


Fig. 11. Dataset statistics for the Tennis and Minecraft datasets.

the y -axis:

$$d_y = \begin{cases} 2r \sin\left(\arccos\left(\frac{r_y}{r}\right)\right) & \text{if } r_y \leq r \\ 0 & \text{otherwise} \end{cases}. \quad (6)$$

Finally, p equals the probability that an interval with size equal to the cross-section, positioned in a random portion of space contained inside an interval of size $d + d_y$, that represents the length of the space that has been touched by the ball while the shutter stays open, contains our point x_c :

$$p(x_c) = \max\left(0, \min\left(\min\left(\frac{d_y}{d}, 1\right), \frac{1}{2} + \frac{d_y}{2d} - \frac{|x_c^y|}{d}\right)\right), \quad (7)$$

where x_c^y is the y -axis coordinate of x_c .

B.2 Racket Modeling

Modeling the scene as a composition of neural radiance fields allows applications such as the insertion of user-defined watertight 3D meshes into the scene. To do so, we first define the 3D bounding

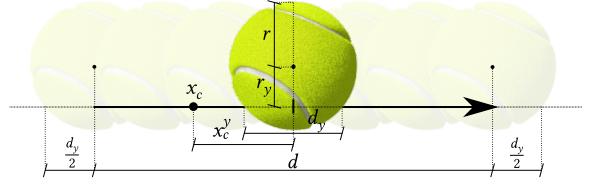


Fig. 12. Visualization of the quantities involved in the computation of the probability p that the ball intersects with a certain point in space during a randomly sampled time instant in the interval from the opening to the closing of the shutter of the camera to capture the current frame. The leftmost and rightmost balls depict the ball position at the times the camera shutter opens and closes respectively. For simplicity, we represent the space where the velocity vector of the ball is aligned with the y -axis.

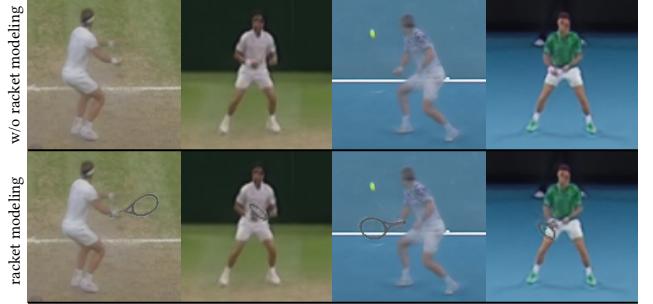


Fig. 13. Examples of tennis scenes with and without inserted rackets.

box for the mesh. Then, we extract the signed distance function (SDF) of the 3D mesh. To allow fast retrieval of SDF values during rendering, we sample SDF values along an enclosing voxel grid so that subsequently they can be efficiently retrieved using trilinear sampling. During neural rendering, when a sampled point intersects with the object's bounding box, we query its SDF function and assign a fixed, high density to points that fall inside the object. For simplicity, we assume the object has a uniform appearance and assign a fixed feature vector to such points. To attach the mesh to an articulated object, we align it to its desired position in the object's canonical space, select which joint the mesh should move according

to, and we modify blending weights W for the desired joint to have a high value in the region corresponding to the mesh (see Eq. (9)).

We employ this technique on the Tennis dataset to manually insert rackets in the scene that cannot be easily learned since they appear frequently blurred and have no ground truth pose available. After the synthesis model is trained, we use this technique to insert a racket mesh in the hand of each player and configure it to move it according to the elbow joint. Fig. 13 shows examples of rackets inserted in tennis scenes.

When inserting additional objects at inference time, we find the enhancer model \mathcal{F} may introduce artifacts at the contours of the inserted object. For this reason, we modify \mathcal{F} with a masking mechanism that directly uses values from the NeRF-rendered RGB image $\tilde{\mathbf{I}}$ before the enhancer rather than the enhanced image $\hat{\mathbf{I}}$ for pixels corresponding to the inserted object and its contour region.

B.3 2D UI Elements

The presence of 2D user interfaces, such as scoreboards, in the training frames may cause artifacts in the final outputs due to attempts of the synthesis model to model these view-inconsistent elements [Menapace et al. 2022]. To address this issue, we assume that the potential regions where such interfaces may be present are known and we never sample training patches that intersect with these regions. In this way, the model does not attempt to generate such UI elements and instead models the underlying portion of the 3D scene using data from different views.

B.4 Skybox Modeling

The Minecraft background is represented as a skybox that is modeled by extending the planar object modeling mechanism of Sec. 3.1.3. In more detail, we sample the feature plane P according to the ray’s yaw and pitch of the current ray, which can be interpreted as querying points on the surface of a sphere with a radius approaching infinity.

C DEFORMATION MODELING

In this section, we present additional details on the deformation model \mathcal{D} used to render articulated objects such as humans. Given an articulated object, we assume its kinematic tree is known and that the transformation $[R_j | \text{tr}_j]$ from each joint $j \in 1, \dots, J$ to the parent joint is part of the object’s properties. From these we can follow the kinematic tree to derive transformations $[R'_j | \text{tr}'_j]$ for each joint from the bounding box coordinate system to the canonical coordinate system. Intuitively, these transformations represent how to map a point x_b in the bounding box coordinate system belonging to the joint j to the corresponding point x_c in the canonical space.

We implement a deformation procedure based on linear blend skinning (LBS) [Lewis et al. 2000] that establishes correspondences between points in the canonical space x_c and in the deformed bounding box space x_b by introducing blending weights w for each point in the canonical space. These weights can be interpreted as the degree to which that point moves according to the transformation

associated with that joint.

$$x_b = \sum_{j=1}^J w_j(x_c) (R'_j x_c - R'^{-1}_j \text{tr}'_j). \quad (8)$$

During volumetric rendering, however, we sample points x_b in the bounding box space and query the canonical volume in the corresponding canonical space point x_c . Doing so requires solving Eq. (8) for x_c , which is prohibitively expensive [Li et al. 2022]. Inspired by HumanNeRF [Weng et al. 2022], instead of modeling LBS weights w , we introduce inverse linear blending weights w^b :

$$w_j^b(x_b) = \frac{w_j(R'_j x_b + \text{tr}'_j)}{\sum_{j=1}^J w_j(R'_j x_b + \text{tr}'_j)}. \quad (9)$$

such that the canonical point can be approximated as:

$$x_c = \sum_{j=1}^J w_j^b(x_b) (R'_j x_b + \text{tr}'_j). \quad (10)$$

We parametrize the function w mapping spatial locations in the canonical space to blending weights as a neural network. Similarly to C , we employ 3D convolutions to map a fixed noise volume $W' \in \mathbb{R}^{F' \times H'_W \times W'_W \times D'_W}$ to a volume of blending weights $W \in \mathbb{R}^{J+1 \times H_W \times W_W \times D_W}$, where each channel represents the blending weights for each part, with an extra weight modeling the background. The volume channels are normalized using softmax, so that they sum to one, and can efficiently be queried using trilinear sampling. To facilitate convergence, we exploit the known kinematic tree to build a prior over the blending weights that increases blending weights in the area surrounding each limb [Weng et al. 2022].

D IMPLEMENTATION DETAILS

D.1 Synthesis Model

We model Minecraft scenes considering as objects the player, the scene, and the background. To model Tennis scenes, we consider as separate objects the two players, the ball, the field plane, and the vertical backplate at the end of the field. Both players share the same canonical representation. Note that the field and backplate are modeled as planar objects due to the lack of camera translation on the tennis dataset, which does not make it possible to reconstruct the geometry of static objects [Menapace et al. 2022].

For each ray, we uniformly sample 32 points for players, 16 for the ball, 48 for the Minecraft scene, and 1 for all remaining objects that are modeled as planes. We do not employ hierarchical sampling, which we empirically found not to improve results. A patch size of 180x180px and of 128x128px are employed respectively for the Tennis and Minecraft datasets.

We model the initial blocks of the style encoder \mathcal{E} as the first two residual blocks of a pretrained ResNet 18 [He et al. 2016]. To prevent players from being modeled as part of the background, we always sample images in pairs from each video and randomly swap the style codes ω of corresponding objects [Menapace et al. 2022].

To represent the player canonical radiance fields, we use a voxel V with $F = 64$ features and $H_V = W_V = D_V = 32$. Deformations are represented using blending weights W with $H_W = W_W = D_W = 32$.

For the Minecraft scene, the size of the voxel V is increased to $H_V = W_V = D_V = 128$. The Minecraft skybox is represented with feature planes P with $F = 64$ features and size $H_P = W_P = 256$. Due to their increased complexity and variety of styles, in the Tennis dataset feature planes P with $F = 512$ features are adopted. The MLPs performing stylization of the canonical field features are modeled using 2 layers with a hidden dimension of 64, with a final number of output features $F = 19$, where the first 3 channels represent radiance.

D.2 Animation Model

For the text encoder, we model \mathcal{T}_{enc} as a frozen T5-Large [Raffel et al. 2022] model and \mathcal{T}_{agg} as a transformer encoder [Vaswani et al. 2017] with 4 layers, 8 heads, and a feature size of 1024. For each sequence, the output a^{emb} of \mathcal{T} is the transformer encoder output corresponding to the position of the end-of-sentence token in the input sequence. We experimented with mean pooling and a learnable class-token with comparable results. Consistently with [Saharia et al. 2022], we found alternative choices for \mathcal{T}_{enc} (T5-Small, T5-Base [Raffel et al. 2022] and the CLIP text encoder [Radford et al. 2021]) to underperform T5-Large.

For the temporal model \mathcal{A} , we employ a transformer encoder with 12 layers, 12 heads, and 768 features. To favor generalization to sequences of different lengths at inference time, we adopt relative positional encodings [Shaw et al. 2018] that specify positional encodings based on the relative distance in the sequence between sequence elements. We produce embeddings for the diffusion timestep k and framerate v using sinusoidal position encodings [Vaswani et al. 2017]. Additionally, to enable the model to better distinguish between noisy sequence entries and conditioning signals, we find it beneficial to condition also on m^s and m^a using the same weight demodulation layer.

The temporal model receives a flattened sequence of object properties grouped and encoded as follows: the position of objects as the bounding box center point; the player poses expressed with joint translations and rotations separately, with rotations expressed in axis-angle representation, which we find to produce more realistic animations with respect to the 6D representation of [Zhou et al. 2019]; the ball speed vector expressed as its orientation in axis-angle representation and norm. Separating positions from joint translations and rotations has the practical implication that these properties can be independently used as conditioning signals during inference. This enables applications such as generating realistic joint rotations and translations given a sequence of object positions in time describing the object movement trajectory. We assume style to remain constant in the sequence, thus we do not include it as input to the model.

E TRAINING DETAILS

E.1 Synthesis Model

We employ a reduced learning rate of $1e - 5$ for the 3D CNNs that model the canonical radiance field voxels V and blending weights W that we find important to improve the learned geometry and avoid artifacts such as holes.

We train our full model on 8 A100 40GB GPUs for 4 days and 2 days respectively on the tennis and Minecraft datasets. We train the reduced version of our model (Ours Small) on 4 A100 40GB GPUs for 3 days and 2 days respectively for the Tennis and Minecraft datasets.

E.2 Animation Model

We create masks m^s and m^a by randomly selecting one of the following masking strategies:

- i randomly mask each sequence element with a probability 0.25
- ii randomly mask each sequence element with a probability 0.5
- iii mask all sequence elements corresponding to a block of consecutive timesteps of random length
- iv the complement of (iii)
- v mask all sequence elements corresponding to the last timesteps of the sequence
- vi mask all sequence elements corresponding to a randomly chosen set of object properties

With probability 0.5, we set $m^a = 1$, excluding actions from the masking operation, so that the model can learn to solve (ii), (iii), (iv) also in the scenario where text guidance is provided. We design the masking strategies to mimic masking configurations that are relevant to inference problems such as autoregressive generation (v), unconditional generation (vi), generating opponent responses to user actions (vi), sequence inpainting (iii), sequence outpainting (iv) and framerate increase (iii).

We train our full model on 8 A100 40GB GPUs for 15 days and 10 days respectively on the tennis and Minecraft datasets. We train the reduced version of our model (Ours Small) on 2 A100 40GB GPUs for 6 days and 4 days respectively for the Tennis and Minecraft datasets.

F INFERENCE DETAILS

F.1 Inference Speed

Our synthesis model renders images at 2.96fps over a single A100 GPU. We can parallelize inference by generating batches of 8 consecutive frames on separate GPUs for 23.7fps. The animation model runs at 1.08fps using 1000 diffusion sampling timesteps. Meng et al. [Meng et al. 2022] show that a reduction to <16 timesteps is possible with no or minimal loss in quality for a projected performance of >67.5fps. Hence, we believe our framework can be made real-time, which is a scope for future works.

F.2 Animation Model Inference Details

At inference time, the user is presented with a fully-masked, empty sequence $s^c = 0$, $m^s = 0$, $a^c = ""$, $m^a = 0$. Any object property can be specified as a conditioning signal in s^c and text action descriptions for any sequence timesteps can be provided in a^c , with masks updated accordingly. The desired framerate v is also specified.

The text encoder \mathcal{T} produces text embeddings a^{emb} as in Eq. (2). Successively, the *reverse process* is started at diffusion time $k = K$, with s_K^p sampled from the normal distribution. The DDPM sampler [Ho et al. 2020] queries the temporal model according to Eq. (3) to

progressively denoise s_k^p and obtain the predicted sequence $s^p = s_0^p$. The final sequence is obtained as $s = s^p + s^c$, following Eq. (1).

F.2.1 High Framerate Generation. To produce sequences at the dataset framerate, we devise a two-stage sampling procedure designed to prevent an excessive increase in the sequence length. In the first stage, we sample the desired sequence at a low framerate v_1 . In the second stage, we exploit the masking mechanism and framerate conditioning to increase the framerate and, consequently, the length of the generated sequence. After the first stage, we consider a higher framerate v_2 and extend the sampled sequence s with new states between existing ones, that we call keyframes, until the sequence length corresponding to v_2 is reached. This sequence constitutes the new s^c . Any previous action conditioning is copied in a new a^c in the corresponding keyframe locations. Masks are updated to be 1 in the position of the keyframes and 0 elsewhere. The sampling process is then repeated with the new conditioning signals and a sequence s is produced at the final framerate v_2 . To avoid an explosion in the length of the sequence, we exploit keyframes to divide the sequence into shorter chunks beginning and terminating at a keyframe, and sampling is performed separately on each chunk.

F.2.2 Autoregressive Generation. Our masking mechanism can be used to produce predictions autoregressively, enabling long sequence generation. Autoregressive generation can be obtained by considering a sequence s^c and removing the states corresponding to the first t timesteps. t timesteps are then added at the end of the sequence and a mask m^s is created to zero out these additional t steps. Conditioning signals can then be specified as desired for the last t timesteps. When sampling s^p , a prediction is produced for the additional timesteps and the procedure can be repeated.

G APPLICATIONS

To demonstrate style swapping capabilities, in Fig. 14 we swap the style of the player ω in the original image with the one from a target image. In addition, our synthesis model renders the current state of the environment from a user-defined perspective, similarly to the rendering component of a game engine. This enables our LGE to perform novel view synthesis as shown in Fig. 15.

H ADDITIONAL EVALUATION

H.1 Robustness to prompt variations

We perform a study on the Tennis dataset to evaluate the capability of our animation model to support diverse language prompts. We randomly sampled 50 prompts from our tennis dataset and asked *ChatGPT* to “Produce a semantically equivalent reformulation of the prompt ‘<prompt>’”. The sentence similarity between original and reformulated prompts measured by Jaccard similarity on their 3-grams is 0.390, while it is 0.203 for random dataset prompt pairs, indicating high diversity. As an example, the prompt “*the player stops and quickly runs to the right and hits the ball with a backhand towards the center of no man’s land*” is reformulated to “*The player comes to a stop and rapidly sprints towards the right before executing a backhand stroke that directs the ball towards the center of no man’s land*”, and prompt “*the player prepares to hit the ball but stops, returning ball hits the net*” is transformed to “*The player readies themselves to strike*

Table 4. Animation model comparison with baselines and ablation on the Tennis dataset. Position and Joints 3D in meters, Root angle in axis-angle representation.

	Position		Root angle		Joints 3D	
	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓
<i>Action conditioned video prediction</i>						
PE	3.117	87.688	1.182	12.627	0.277	30.711
Rec. LSTM	1.753	7.413	1.100	8.416	0.234	18.455
Rec. Transf.	1.183	2.996	0.913	7.566	0.212	15.976
Ours Small	1.244	1.071	1.187	0.601	0.178	1.570
Ours	1.064	0.846	0.961	0.421	0.153	1.049
<i>Unconditional video prediction</i>						
PE	3.973	146.019	1.604	30.448	0.437	78.835
Rec. LSTM	2.064	11.283	1.224	14.860	0.264	28.736
Rec. Transf.	1.649	10.514	1.123	15.648	0.251	27.258
Ours Small	2.352	2.271	1.455	0.781	0.213	1.827
Ours	1.925	1.377	1.277	0.518	0.192	1.261
<i>Opponent modeling</i>						
PE	1.720	40.766	0.814	6.783	0.246	40.441
Rec. LSTM	0.990	4.809	0.606	2.411	0.132	9.305
Rec. Transf.	0.294	0.364	0.403	1.623	0.100	5.628
Ours Small	0.344	0.187	0.581	0.301	0.088	0.765
Ours	0.252	0.143	0.437	0.198	0.069	0.478
<i>Sequence completion</i>						
PE	4.353	641.976	0.903	13.955	0.251	62.981
Rec. LSTM	1.581	5.507	0.697	2.517	0.143	10.443
Rec. Transf.	1.169	3.735	0.631	2.514	0.138	10.519
Ours Small	1.578	2.243	0.832	0.560	0.114	0.851
Ours	1.153	1.349	0.703	0.288	0.101	0.558
<i>Average</i>						
PE	3.291	229.112	1.126	15.953	0.303	53.242
Rec. LSTM	1.597	7.253	0.907	7.051	0.193	16.735
Rec. Transf.	1.074	4.402	0.767	6.838	0.175	14.845
Ours Small	1.380	1.443	1.014	0.560	0.148	1.253
Ours	1.099	0.929	0.844	0.356	0.129	0.836

the ball, but abruptly halts and as a result, the returned ball collides with the net”.

Successively, we run an AMT user study. Users are shown a video and two prompts (the true prompt used to produce the video, and a random negative prompt) and they are asked to recognize which of two prompts is the one used to produce a certain video. The average accuracy over 500 responses is 74.4% and 77.1% for videos produced using reformulated and dataset prompts respectively, indicating capability of the model to generate videos matching prompts independently from the form of used language.

The model trained on Minecraft allows for limited prompt variation due to the synthetic nature of the training language whose limited variation does not enable the model to learn generalization capabilities to different sentence structures as the ones acquired for Tennis. This limitation could be addressed by improving the synthetic language generation process, which we leave as future work.

H.2 Animation Model Evaluation

In Tab. 4 and Tab. 5 we show evaluation results for each inference task respectively on the Tennis and Minecraft datasets. In Fig. 16 we show qualitative results on the Minecraft dataset.

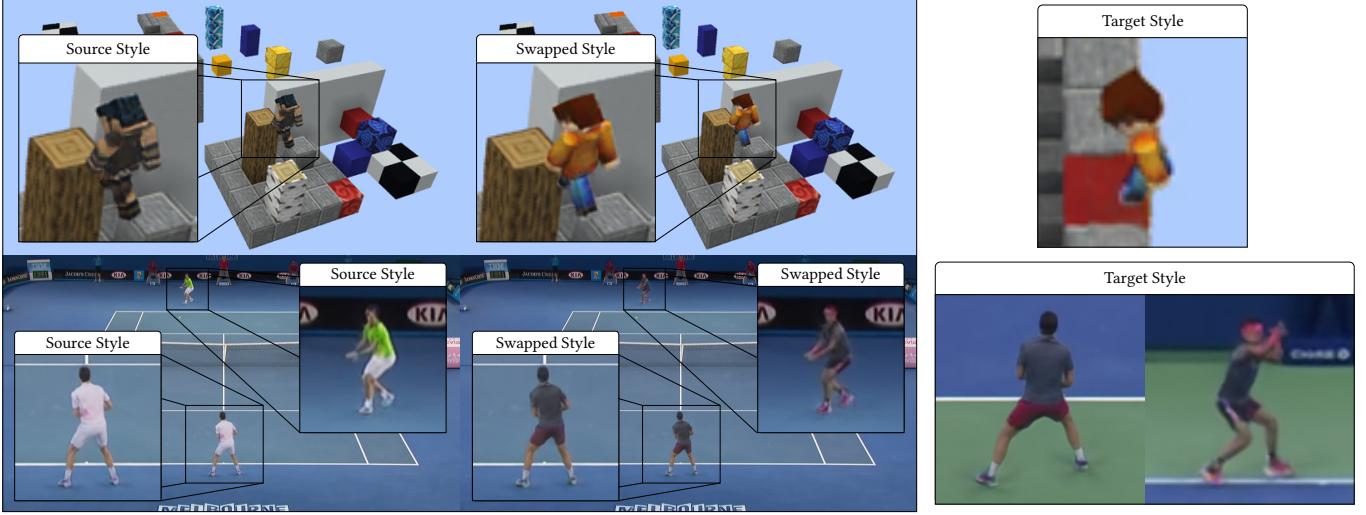


Fig. 14. Style swap results on the Tennis and Minecraft datasets. We produce the central image by swapping the style code ω for the players on the leftmost image with the ones from the rightmost image. Minecraft results are cropped for better visualization.

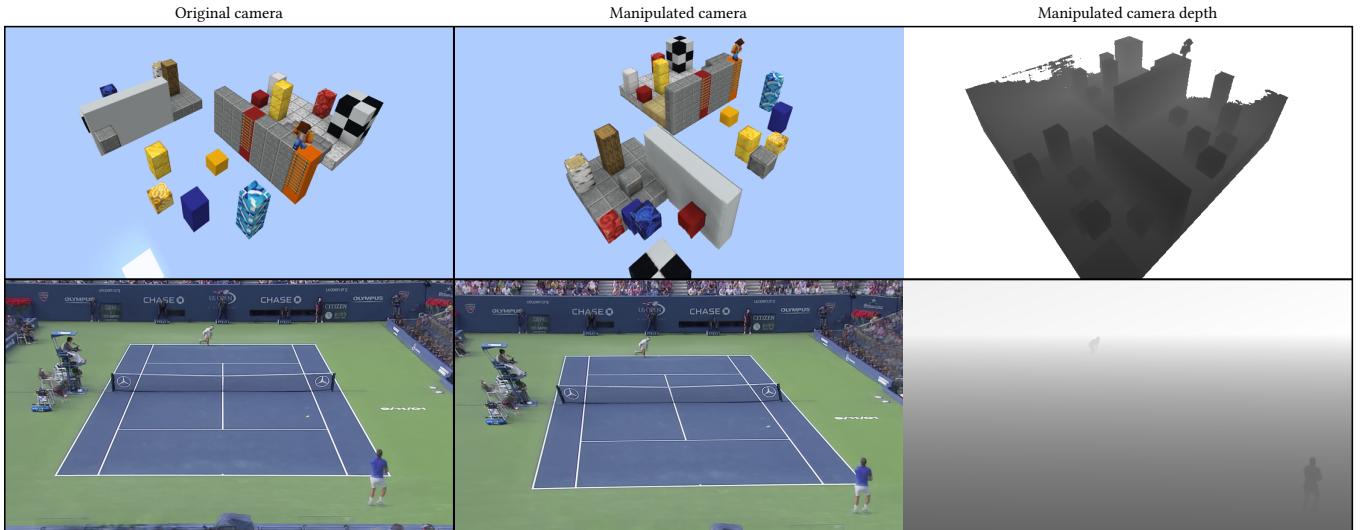


Fig. 15. Camera manipulation results on the Tennis and Minecraft datasets. The lack of camera translation on the Tennis dataset does not allow to capture the 3D geometry of static objects, which is replaced by a prior [Menapace et al. 2022]. Minecraft results are cropped for better visualization. Additional video examples are shown on the *Supp. Website*.

H.3 Alternative Samplers

In this section, we evaluate our animation model using the DDIM [Song et al. 2021] sampler with a varying number of timesteps and show results in Fig. 17. The DDIM sampler produces samples with a lower number of sampling timesteps with respect to the DDPM [Ho et al. 2020] sampler at the cost of higher FD scores, thus providing a tradeoff between inference speed and sample quality. We note that several techniques exist that speed up diffusion model sampling [Meng et al. 2022; Salimans and Ho 2022] and that these efforts are orthogonal to our work.

I DISCUSSION

I.1 Cost-Quality Tradeoff and Game Developers Validation

Building videogames is an extremely expensive process. We propose a *fully learnable* solution, requiring only annotated monocular videos and supporting the core set of game functions.

This research direction is a step towards creating games and editing videos without expensive equipment, data, 3D assets, sophisticated software and manual labor of trained experts. We do not aim to surpass the quality of high-cost techniques (\$100k-\$1M, see below) that require such resources.

Table 5. Animation model comparison with baselines and ablation on the Minecraft dataset. Position and Joints 3D in meters, Root angle in axis-angle representation.

	Position		Root angle		joints 3D	
	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓
<i>Action conditioned video prediction</i>						
PE	2.720	90.904	1.822	23.949	0.365	47.956
Rec. LSTM	2.623	54.927	2.040	62.363	0.579	118.592
Rec. Transf.	2.798	76.582	1.794	52.677	0.506	100.731
Ours Small	0.533	2.494	0.901	5.624	0.145	4.083
Ours	0.523	2.582	0.749	4.578	0.135	3.794
<i>Unconditional video prediction</i>						
PE	3.994	197.434	2.111	59.112	0.370	46.665
Rec. LSTM	2.850	68.915	1.999	69.886	0.581	121.187
Rec. Transf.	2.834	76.780	1.795	50.871	0.480	87.584
Ours Small	2.341	9.795	1.814	8.969	0.199	4.422
Ours	2.330	11.032	1.685	5.648	0.197	4.385
<i>Sequence completion</i>						
PE	1.504	29.582	0.926	10.634	0.197	24.096
Rec. LSTM	1.401	18.044	1.066	17.664	0.309	59.750
Rec. Transf.	0.830	6.232	0.700	4.822	0.170	21.615
Ours Small	0.379	1.095	0.516	3.456	0.077	2.265
Ours	0.343	0.830	0.433	2.022	0.065	1.899
<i>Average</i>						
PE	2.739	105.973	1.620	31.232	0.311	39.572
Rec. LSTM	2.292	47.296	1.702	49.971	0.489	99.843
Rec. Transf.	2.154	53.198	1.430	36.123	0.385	69.977
Ours Small	1.084	4.461	1.077	6.016	0.140	3.590
Ours	1.065	4.815	0.956	4.083	0.132	3.360

Evaluation of such a method necessarily needs to be performed under the light of a Pareto curve representing the tradeoff between development cost and output quality. We analyze three points on this curve for tennis:

- *Traditional game development* (1M\$ cost range, high quality). We interviewed three game development experts with backgrounds in real-time graphics, 3D models and animation, and game development management with 45 years of combined experience. We invited the experts to discuss the recent “AO Tennis 2” game and compare it with our method. Their cost estimate for building AO Tennis 2 using existing game engines were respectively of \$100k-500k (in the US), \$600k (45-person-years in Ukraine), and of \$1M (3-person-years in the US), including software licenses, equipment and assets. They noted that “[our model’s] game AI is very valuable and it’s going to be the hardest part of developing tennis” and that our model’s game AI has the potential to be “game changer” for tasks such as realistically modeling the behavior of animals inserted in a game. With regards to graphics, they noted that “[our model’s graphics] is more realistic” and that “[our model’s output] looks like a real video” when not zoomed in, but commented that users may prefer a less-realistic game-like graphics because they are more accustomed to its look. The users highlighted the value of the generated animated 3D assets, remarking that high-quality 3D assets of real players are expensive. When asked about possible immediate uses of the model in game production they reported that while

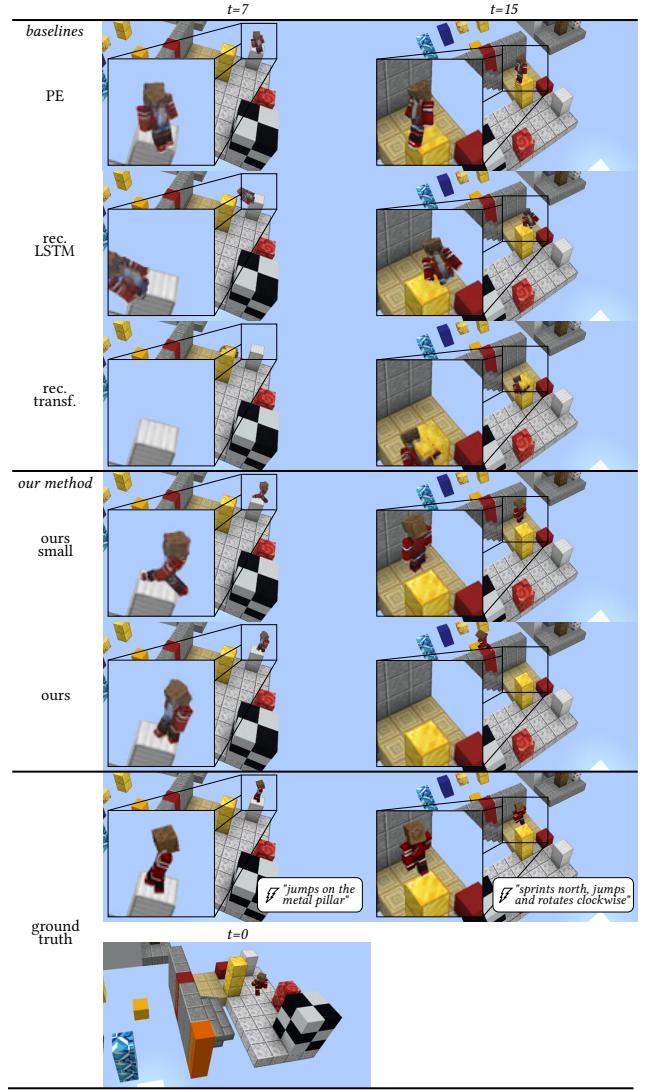


Fig. 16. Qualitative results on the Minecraft dataset. Sequences are produced in a video prediction setting that uses the first frame object properties and all actions as conditioning. Additional samples are shown in the *Supp. Website*.

the model is “impressive” it is not yet “mind-blowing” when speaking about building products using our method. In particular, the model’s output may present artifacts that would require correction, preventing direct use of the model in a production environment. An interviewee highlighted that the framework could be called a “limited game engine” and would be “awesome” to use to create a new type of “promo games” such as Superbowl or Nascar games that can be released at low cost immediately after the sport season, leveraging the captured footage. Given the significantly lower cost of our method and the rapid evolution in neural rendering and diffusion models on which our framework is based, we consider

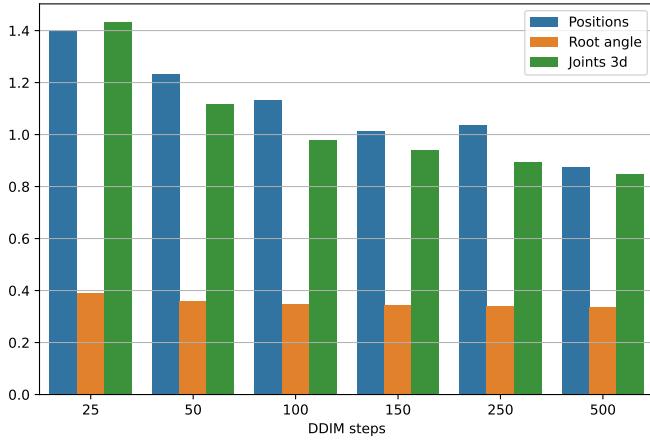


Fig. 17. Evaluation results for our method on the Tennis dataset using DDIM sampler with a varying number of sampling steps.

these comments an encouraging validation of this research direction.

- *Specialized CG techniques* (100k\$ cost range, high quality). Specialized character animation techniques [Holden et al. 2020; Starke et al. 2019, 2020] produce high-quality animations. However, they come at higher cost. The sole requirement of motion capture data entails a professional multicamera system (1k-10k\$ per camera * 10s of cameras), motion capture software licenses (1k-10k\$/year), an HPC system with TBs of storage (>10k\$), dedicated engineers (10k-100k\$/year), studio space, inviting professional actors or players (10-100\$/h). In addition, they do not model the complete game’s logic and only learn basic game AI elements, making their integration into a unified, learnable framework nontrivial.

I.2 Limitations

Since the model is trained on a dataset showing only plausible actions, the model’s behavior is not defined when an *implausible* action is specified, such as hitting a ball when it is too far from the player or jumping on a platform that is out of reach. In these cases we find the model to produce outcomes such as running too fast or producing implausibly long jumps. In addition, the model does not generate plausible results in the case of actions extremely out of distribution such as performing a backflip or doing a push-up.

While our Tennis dataset contains varied text annotations that allow the model to generalize to text inputs with varied structure, our Minecraft dataset’s synthetic text annotations are less varied and the fixed synthetic structure of sentences tends to be memorized, making the model less effective if a different syntax is used (see Sec. H.1). To address this issue, a more sophisticated algorithm can be employed to generate action annotation on the Minecraft dataset.

Our model learns to associate referential language to scene coordinates rather than the appearance of the referred object, and the model memorizes the position of contact surfaces. While tennis scenes always have the same structure, for Minecraft the model cannot generalize to different scenes. This concern can be addressed by

conditioning the animation model on the scene’s geometry, which we leave as future work.

Our animation model outperforms baselines that operate under the same data assumptions [Menapace et al. 2022] in terms of animation quality. With respect to recent character animation methods [Holden et al. 2020; Starke et al. 2019, 2020] making use of richly annotated motion capture data and dataset-specific handcrafted optimizations (see Sec. 2.4), our method demonstrates more advanced game logic and game AI modeling capabilities, but produces foot sliding artifacts. We expect continuous improvements in diffusion models to alleviate such artifacts and expect further improvements by considering different parametrizations of pose parameters taking into consideration the distance of limbs from the terrain, which we will explore in future work.

Lastly, our animation model does not yet produce results in real-time. We discuss inference speed and strategies to make the model real-time in Appx. F.1. Improving the sampling speed of diffusion models is an actively investigated problem [Meng et al. 2022; Salimans and Ho 2022; Song et al. 2021] that is orthogonal to ours.

I.3 Ethics

The techniques described in this work fall in the category of video editing methods and could potentially be used to nefariously alter existing videos. The design of our method assumes multiple camera-calibrated observations of a single scene to be available for training, and that the desired edit is shown at least once in the training data. This provides protection towards applying the method to tamper a single video, for which the quantity of data would not be sufficient and the desired edit would likely not be shown.