Department of Information Engineering and Computer Science
IECS Doctoral School

Towards Intelligent Videogame Generation

Videos to Games using Deep Learning

A thesis submitted for the degree of
Doctor of Philosophy

Advisor
Elisa Ricci

Co-Advisor
Sergey Tulyakov

Reviewers
Vladislav Golyanik
Stéphane Lathuilière

Student
Willi Menapace

Academic year 2023/2024

Acknowledgements

My PhD studies have been supported by a number of extraordinary people towards which goes my deepest gratitude. Elisa and Sergey for their expert guidance, always carried out with empathy and focused on my personal growth. Stéphane for patiently passing to me his research methodology and academic writing knowledge. Aliaksandr for his close supervision, frequent research discussion and material help. Vlad and Christian for welcoming me in their research group and sharing their computer graphics expertise. Elena and all my family for their unlimited support which has been the foundation for sustaining my studies.

Abstract

Multimedia content plays a pivotal role in contemporary society, permeating diverse facets of communication and entertainment. However, the creation of high-quality multimedia content often requires specialized expertise and equipment, limiting its widespread accessibility. To address these challenges, deep learning-based methods have emerged as revolutionary tools for image and video generation, reshaping content creation landscapes and transforming professional workflows. These techniques have significantly impacted the media creation industry, facilitating more accessible and efficient media pipelines. Nevertheless, videogame development remains a resource-intensive endeavor.

Videogames are complex soft real-time systems that necessitate meticulous modeling of 3D objects, animations, graphical effects, physics, and intelligent agents. The collaborative efforts of various specialized professionals, including designers, software engineers, 3D artists, and sound engineers, are required to develop videogames, each contributing their expertise to different aspects of the game. Furthermore, the demanding quantity of assets involved often requires specialized hardware such as 3D and motion capture equipment, professional cameras, and sound recording devices. Although early attempts to democratize game development introduced game engines to provide a strong foundation for the development process, creating videogames remains a multi-million-dollar endeavor even with sophisticated modern engines. This raises the question of whether deep learning can revolutionize videogame creation in a similar manner.

This research proposes several novel methods aimed at streamlining the videogame creation process, leveraging the abundance of available videos showcasing gameplay. These methods automatically convert readily accessible video collections into playable game experiences.

Firstly, we introduce a completely unsupervised pipeline called Playable Video Generation, which seamlessly transforms raw video collections into playable experiences. We demonstrate the effectiveness of this pipeline through compelling applications, including Atari games emulation, robotic arm control, and tennis simulations. Recognizing that videogames predominantly involve rendered compositions of 3D objects rather than 2D videos, we extend the framework to 3D environments in Playable Environments. Here, objects are represented as separate 3D entities, allowing for control over multiple entities and their attributes, such as style, as well as camera control. Lastly, we present the concept of Promptable Game Model, where intelligent agents within the game are empowered through the use of natural language instructions. This integration enables the creation of complex playable experiences in which agents are controlled through fine-grained actions and can act independently based on their own "game AI" to defeat opponents and navigate intricate environments.

By harnessing the power of deep learning techniques, this research aims to democratize videogame development by enabling the transformation of annotated video collections into playable experiences. The proposed methods provide a promising avenue for revolutionizing the creation of videogames, similar to the transformative impact deep learning has had on the media creation industry.

To facilitate future work in this novel area, all the data and code is made publicly available.

Contents

Chapter 1

Introduction

1.1 Motivation

Multimedia content, encompassing images, videos, and audio, holds a central role in contemporary society, serving as a powerful tool for communication, entertainment, and information sharing. The ubiquity of social media platforms, the interactions between companies and customers, and the engagement between governments and citizens are all reliant on multimedia content. However, the creation of professional-quality multimedia content is a complex and demanding endeavor that requires specialized knowledge, manual labor, and professional equipment.

The emergence of deep learning techniques is bringing about a revolutionary impact on content creation, streamlining the process and transforming professional tools. This has brought to the integration of deep learning models into widely used software suites, such as Adobe Creative Suite, enabling professionals to create high-quality media with unprecedented ease. Notable advancements [41, 43, 111, 114, 138, 148], have demonstrated the potential of deep learning to enable users without specialized knowledge to create high-quality multimedia content.

While traditional multimedia has primarily focused on passive consumption, the ubiquitous access to powerful devices coupled with the advent of augmented reality (AR) and virtual reality (VR) technologies, has given rise to a growing presence of interactive multimedia experiences. These interactive experiences have gained prominence due to their enhanced engagement potential, making them more effective means of communication and entertainment. Videogames, with their inherent emphasis on user interaction and engagement, have traditionally embodied this interactive multimedia paradigm. As the multimedia industry undergoes a potential shift toward increased interactivity, the importance of videogame creation becomes even more pronounced.

Despite their significance, the development of videogames remains a highly demanding process comparable in its sophistication and used tools to that of top movie production [35]. This is due to the collaborative efforts of game studios comprising various trained experts such as 3D artists, software engineers, animators, game designers and sound engineers, as well as the requirement of specialized equipment such as professional cameras and sound recording devices, motion capture and 3D scanning equipment. Well-known videogame projects, such as AAA titles, exemplify the substantial financial investments associated with achieving the desired level of quality. Over the years, the industry has endeavored to streamline the game development process. The introduction of game engines provided a solid foundation for the development efforts, while the advancements in technologies such as 3D object capture and motion capture brought to improvement in the asset creation process. However, even with these innovations, the creation of videogames remains complex, as the expectations for quality continue to rise.

To date, deep learning techniques have only been harnessed to a limited extent within the realm of videogame creation [56, 128, 151]. Existing learning-based approaches for game development have predominantly focused on specific aspects of games, such as the reconstruction of generation of 3D assets [64, 92, 96] or character animation [8, 45, 71, 128, 129]. While these advancements have demonstrated their efficacy in narrow domains of videogame creation, they lack a holistic approach that could replace the entire game development pipeline. Consequently, the democratization of videogame creation, in which playable experiences can be generated by leveraging deep learning techniques without substantial human effort, remains an open important problem.

In light of these challenges and opportunities, this work aims to address the need for a streamlined and accessible method to create complex interactive multimedia experiences. We propose several techniques that make a first step towards the automated generation of playable experiences and hope advancements in this research direction will bring advancements to the videogame creation industry, akin in magnitude to the transformative impact deep learning is having on the broader media creation industry.

1.2 Outline

The foundation of this research is built upon the key observation that there is a wealth of readily available videos showcasing gameplay. Leveraging this vast collection of videos, we propose to harness their potential to train a deep learning model that enables users to generate video outputs interactively, with a high degree of control over every aspect of the generation process. This level of control is a fundamental characteristic that distinguishes the task of creating playable experiences from unconditional video generation.

The thesis is structured as follows. In Chapter 2, we present the Playable Video Generation framework, a completely unsupervised pipeline that transforms collections of videos into playable experiences, where users can control a player using a set of discrete actions. We demonstrate the effectiveness of this pipeline through compelling applications, including Atari games emulation, robotic arm control, and tennis simulations. Moving forward, in Chapter 3, we recognize that videogames predominantly involve rendered compositions of 3D objects rather than 2D videos. To address this, we extend the framework to 3D environments, introducing the Playable Environments framework. In Playable Environments, objects are represented as independent 3D entities, each possessing its own properties, such as style, which can be rendered in arbitrary compositions from user-defined cameras. This extension enables independent control of multiple agents and the camera within the game environment. Finally, Chapter 4 introduces Promptable Game Models, where we enhance the level of control over the playable experience by integrating natural language instructions. This integration empowers intelligent agents within the game to respond to fine-grained actions and act independently based on their own "game AI". As a result, complex playable experiences can be created, allowing agents to defeat opponents and navigate intricate environments.

1.2.1 Playable Video Generation

In Chapter 2, we introduce a novel task called Playable Video Generation (PVG), that mimics the ability of humans to anticipate actions and interactions in videos without supervision. The task of PVG aims to learn a set of distinct actions from real-world video clips in an unsupervised manner and to generate videos conditioned on such actions, enabling users to interactively generate new videos, providing an experience similar to a videogame. Unlike existing future frame prediction methods that condition generation on a single action label [58, 140], PVG allows users to provide discrete action labels at each time step during video generation, a critical component of interactivity.

The chapter presents a framework named Clustering for Action Decomposition and Discovery (CADDY) for PVG, that employs an encoder-decoder architecture with a discrete bottleneck layer and an action network to capture both discrete and continuous aspects of actions.

Experimental results demonstrate that CADDY produces high-quality videos and offers an improved playability experience compared to existing methods. The datasets used for evaluation include real-world videos (such as tennis and robotics) as well as synthetic video game data based on Atari videogames.

1.2.2 Playable Environments

Chapter 3 introduces the concept of Playable Environments (PE) as a means to enhance the complexity and user control in modeled playable experiences. This chapter recognizes that real scenes can be more accurately represented as a collection of independent 3D objects, aligning with the prevalent paradigm adopted in game engines. This representation allows for various applications, including the independent control of players through discrete actions, the independent control of object attributes such as style, and explicit control over the camera. With respect to PVG, building such representation requires supervision in the form of calibrated cameras and 3D object detection but enables a much wider degree of control.

The proposed method adopts a composite state representation, which describes the position and attributes of all objects within the scene. It introduces a synthesis module based on a compositional Neural Radiance Fields (NeRF) framework, enabling the rendering of the environment state from a controllable camera perspective. Additionally, an action module is developed to facilitate playability by modeling the actions and dynamics for each object in the scene.

To evaluate the effectiveness of the proposed method, two large-scale datasets are utilized: one synthetic and one real. The evaluation demonstrates the method's ability to generate high-quality videos with superior playability, camera control, and video quality compared to existing baseline approaches. Furthermore, the introduction of these challenging datasets aims to encourage further research and advancements in this research direction.

1.2.3 Promptable Game Models

Chapter 4 presents an investigation into Promptable Game Models (PGMs), which are similar to PEs in that they learn 3D representations of modeled videogames, wherein each object and agent is portrayed as an independent entity with a unique set of attributes expressed through a state representation. However, PGMs differentiate themselves from PEs by introducing a more detailed control over agents and empowering them with "Game AI" capabilities that enable intelligent behavior within the game environment, such as defeating opponents or navigating scenes proficiently. The key insight for enabling these capabilities lies in leveraging natural language as a means to impart agents with an understanding of actions and their meanings, acting as a form of fine-grained conditioning to specify the behavior of the agents.

The proposed method consists of two primary components. Firstly, a synthesis module, constructed using a composition of Neural Radiance Fields (NeRFs), learns to render an interpretable state representation to a frame. This representation encompasses all attributes of the objects present in the scene. Secondly, a masked animation module takes in conditioning signals in the form of a partial sequence of state representations and a sparse sequence of textual action inputs, which represent actions that should be performed by the agents. The module generates a complete sequence of states as its output. The conditioning inputs establish constraints that the generated sequence must adhere to. Various inference modalities are supported by modifying the conditioning signals, allowing for different interactions. For example, if no actions or states are specified for the opponent, the user can play against an automatically generated opponent. The animation model is built upon a diffusion framework to generate realistic sequences of states.

To facilitate the learning of videogames from videos and textual inputs, two extensive datasets are introduced. These datasets consist of videos accompanied by textual descriptions of the actions performed by each agent within the scene. The evaluation of the proposed model demonstrates its superiority over previous state-of-the-art methods. Additionally, the model enables a range of compelling applications, such as engaging in matches against opponents, generating fictional matches, comprehending the actions required to defeat an opponent, and performing scene navigation based on specified waypoints.

1.3 Contributions

- **Unsupervised Playable Video Generation Framework** We propose a novel framework for generating playable video experiences from collections of videos without the need for manual supervision. The framework is based on a a novel combination of a discrete bottleneck with a latent space clustering procedure to uncover a meaningful representation of actions performed by the agent in the videos. This procedure disentangles high-level actions and action variations, allowing for interactive video generation conditioned on the high-level actions.

- **Playable Environments as Controllable 3D Environment Representations** We present a framework for constructing controllable 3D scene representations named Playable Environments. We introduce a composable Neural Radiance Fields (NeRF) representation to represent scenes as compositions of 3D objects, each with its own attributes. This novel representation enables independent control over the attributes of the 3D objects, contributing to highly customizable and interactive environments that offer the flexibility to manipulate the camera angles, styles, and actions of the agents.

- **Promptable Game Models as Learnable Game Representations** We introduce Promptable Game Models, learnable representations of games offering a subset of capabilities of modern game engines. These engines model controllable 3D environments populated by intelligent agents, that can act independently or be controlled through a set of constraints, expressed as a combination of high-level textual prompts or target states of the agents. The key to achieve this is a novel masked diffusion transformer conditioned on text and target environment states that encapsulates "Game AI" to evolve the environment according to intelligent agent behavior.

- **Evaluation Protocol and Synthesis Quality Metrics** To assess the quality of synthesized scenes featuring controllable agents, we devise an evaluation protocol comprising a set of well-defined metrics. This evaluation protocol serves as a standardized approach to gauge the performance and effectiveness of the proposed methods in the context of the research.

- **Datasets and Resources for Further Studies** We contribute a series of datasets to facilitate future research in this domain. Specifically, we present a large-scale 15-hour dataset of Tennis videos with precise camera calibration, 3D ball tracking, and 3D body pose estimations for the players. Furthermore, each agent's actions in every frame are richly described through textual captions. Additionally, we provide Minecraft datasets of videos with corresponding synthetic annotations. Moreover, we release the necessary code to acquire annotated sequences directly from the Minecraft game engine. These datasets offer a unique combination of data for studying the challenges associated with generating videogames and enhancing the development of related methodologies.

1.4 Publications

In the following list, we give an overview of the publications authored during the PhD, with entries marked with an * not being discussed in this manuscript:

- Willi Menapace, Aliaksandr Siarohin, Ivan Skorokhodov, Ekaterina Deyneka, Tsai-Shien Chen, Anil Kag, Yuwei Fang, Aleksei Stoliar, Elisa Ricci, Jian Ren, Sergey Tulyakov. "Mind the Time: Scaled Spatiotemporal Transformers for Text-to-Video Synthesis". Paper currently under submission.*

- Tsai-Shien Chen, Aliaksandr Siarohin, Willi Menapace, Ekaterina Deyneka, Hsiang-wei Chao, Byung Eun Jeon, Yuwei Fang, Hsin-Ying Lee, Jian Ren, Ming-Hsuan Yang, Sergey Tulyakov. "Panda-70M: Captioning 70M Videos with Multiple Cross-Modality Teachers". Paper currently under submission.*

- Luca Zanella, Willi Menapace, Massimiliano Mancini, Yiming Wang, Elisa Ricci. "Harnessing Large Language Models for Training-free Video Anomaly Detection". Paper currently under submission.*

- Anil Kag, Junli Cao, Willi Menapace, Aliaksandr Siarohin, Sergey Tulyakov, Jian Ren. "AsCAN: Asymmetrically Distributed Convolution-Attention Neural Networks". Paper currently under submission.*

- Ivan Skorokhodov, Willi Menapace, Aliaksandr Siarohin, Sergey Tulyakov. "Hierarchical Patch-wise Diffusion Models for High-Resolution Video Generation". Paper currently under submission.*

- Nicola Dall'Asen, Willi Menapace, Elia Peruzzo, Enver Sangineto, Yiming Wang, Elisa Ricci. "Collaborative Neural Painting". Paper currently under submission.*

- Luca Zanella, Benedetta Liberatori, Willi Menapace, Fabio Poiesi, Yiming Wang, Elisa Ricci. "Delving into CLIP latent space for Video Anomaly Recognition". Paper currently under submission.*

- Chieh Hubert Lin, Hsin-Ying Lee, Willi Menapace, Menglei Chai, Aliaksandr Siarohin, Ming-Hsuan Yang, Sergey Tulyakov. "InfiniCity: Infinite-Scale City Synthesis". In ICCV 2023.*

- Elia Peruzzo, Willi Menapace, Vidit Goel, Federica Arrigoni, Hao Tang, Xingqian Xu, Arman Chopikyan, Nikita Orlov, Yuxiao Hu, Humphrey Shi, Nicu Sebe, Elisa Ricci. "Interactive Neural Painting". In CVIU, 2023.*

- Willi Menapace, Aliaksandr Siarohin, Stéphane Lathuilière, Panos Achlioptas, Vladislav Golyanik, Sergey Tulyakov, Elisa Ricci. "Promptable Game Models: Text-Guided Game Simulation via Masked Diffusion Models". In ACM Transactions On Graphics. Chapter 4 is mainly based on this publication.

- Aliaksandr Siarohin, Willi Menapace, Ivan Skorokhodov, Kyle Olszewski, Hsin-Ying Lee, Jian Ren, Menglei Chai, Sergey Tulyakov. "Unsupervised Volumetric Animation". In CVPR 2023.*

- Matteo Farina, Luca Magri, Willi Menapace, Elisa Ricci, Vladislav Golyanik, Federica Arrigoni. “Quantum Multi-Model Fitting”. In CVPR 2023 (Highlight).*
- Federica Arrigoni, Willi Menapace, Marcel Seelbach Benkner, Elisa Ricci, Vladislav Golyanik. “Quantum Motion Segmentation”. In ECCV 2022.*
- Willi Menapace, Stéphane Lathuilière, Aliaksandr Siarohin, Christian Theobalt, Sergey Tulyakov, Vladislav Golyanik, Elisa Ricci. “Playable Environments: Video Manipulation in Space and Time”. In CVPR 2022. Chapter 3 is mainly based on this publication.
- Willi Menapace, Stéphane Lathuilière, Sergey Tulyakov, Aliaksandr Siarohin, Elisa Ricci. “Playable Video Generation”. In CVPR 2021 (Oral presentation). Chapter 2 is mainly based on this publication.
- Willi Menapace, Stéphane Lathuilière, Elisa Ricci. “Learning to Cluster under Domain Shift”. In ECCV 2020.*
- Subhankar Roy, Willi Menapace, Sebastiaan Oei, Ben Luijten, Enrico Fini, Cristiano Saltori, Iris Huijben, Nishith Chennakeshava, Federico Mento, Alessandro Sentelli, Emanuele Peschiera, Riccardo Trevisan, Giovanni Maschietto, Elena Torri, Riccardo Inchingolo, Andrea Smargiassi, Gino Soldati, Paolo Rota, Andrea Passerini, Ruud JG Van Sloun, Elisa Ricci, Libertario Demi. “Deep learning for classification and localization of COVID-19 markers in point-of-care lung ultrasound”. In TMI, 2020.*

Chapter 2

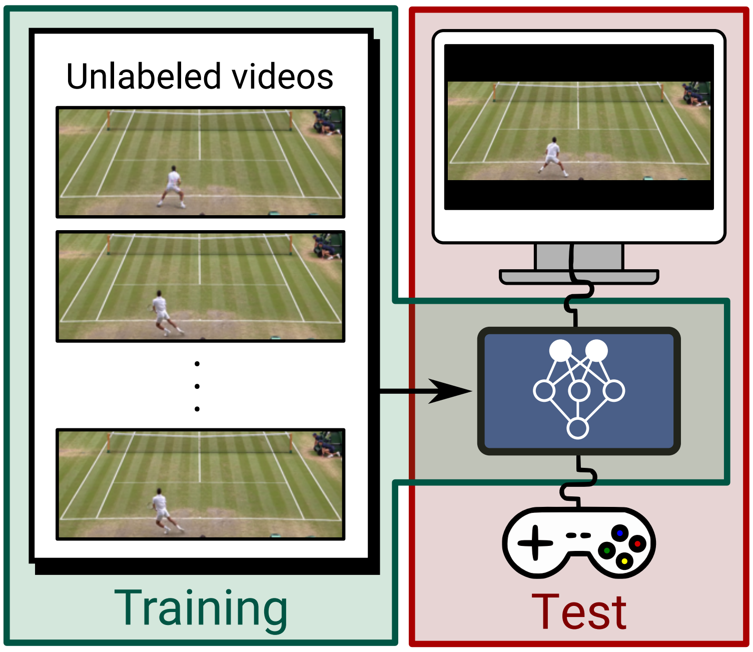
Playable Video Generation

In this chapter we begin our study of the problem of generating playable experiences from videos by introducing the problem of playable video generation (PVG). To enable application of the method beyond synthetic scenarios, a limitation of concurrent methods [16, 56, 98], we define PVG as an unsupervised problem, making it the first method for playable experiences generation to be demonstrated on non-synthetic video datasets. In PVG, we aim at allowing a user to control the generated video by selecting a discrete action at every time step as when playing a video game. The choice of discrete actions lies in their ease of use in that they can directly be mapped to controller buttons, as opposed to continuous action representations that are difficult for users to select and may require supervision in mapping them to controller buttons [113]. The difficulty of the task lies both in learning semantically consistent actions that allow control of all the main aspect of the game and in generating realistic videos conditioned on the user input. We propose a novel framework for PVG that is trained in a self-supervised manner on a large dataset of unlabelled videos. We employ an encoder-decoder architecture where the predicted action labels act as bottleneck. A novel action representation is introduced that separates the high-level discrete component representing the semantics of the action from the low-level continuous action component expressing the variation of the particular action. The network is constrained to learn a rich action space using, as main driving loss, a reconstruction loss on the generated video. We demonstrate the effectiveness of the proposed approach on several datasets, both real and synthetic, with wide environment variety. To facilitate further research in this direction, we make code and data publicly available.

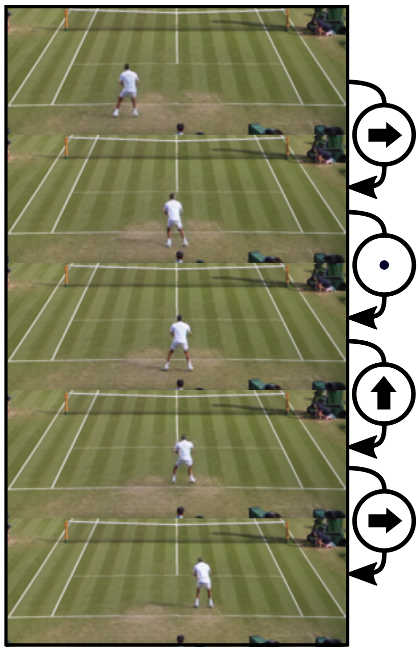
2.1 Introduction

Humans at a very early age can identify key objects and how each object can interact with its environment. This ability is particularly notable when watching videos of sports or games. In tennis and football, for example, the skill is taken to the extreme. Spectators and sportscasters often argue which action or movement the player should have performed in the field. We can understand and anticipate actions in videos despite never being given an explicit list of plausible actions. We develop this skill in an unsupervised manner as we see actions live and on the screen. We can further analyze the technique with which an action is performed as well as the "amount" of action, i.e. how much to the left. Furthermore, we can reason about what happens if the player took a different action and how this would change the video. Learning a video generator in an unsupervised manner with such action understanding abilities would greatly enhance the level of semantic control achievable in the video generation process, enabling users to interactively generate and edit videos given desired actions.

From this observation, we propose a new task, Playable Video Generation (PVG) illustrated in Fig 2.1a. In PVG, the goal is to learn a set of distinct actions from real-world video clips in



(a) Playable Video Generation



(b) Our results

Figure 2.1: We introduce the task of playable video generation in an unsupervised setting (left). Given a set of unlabeled video sequences, a set of discrete actions are learned in order to condition video generation. At test time, using our method, named CADDY, the user can control the generated video on-the-fly providing action labels.

an unsupervised manner (green block) in order to offer the user the possibility to interactively generate new videos (red block). PVG considers learning a set of discrete actions as they can directly be mapped to controller buttons, giving users an intuitive mean to control video generation, as opposed to continuous actions that require more complex mappings [113]. As shown in Fig 2.1b, at test time, the user provides a discrete action label at every time step and can see live its impact on the generated video, similarly to video games. Introducing this novel problem paves the way toward methods that can automatically simulate real-world environments and provide a gaming-like experience.

PVG is related to the future frame prediction problem [29, 70, 83, 126, 135, 139] and in particular to methods that condition future frames on action labels [57, 97, 98]. Given one or few initial frames and the labels of the performed actions, such systems aim at predicting what happens next in the video. For example, this framework can be used to imitate a desired video game using a neural network with a remarkable generation quality [57,98]. However, at training time, these methods require videos with their corresponding frame-level action at every time step. Consequently, these methods are limited to video game environments [57, 98] or robotic data [97], where such annotation can be automatically generated, and cannot be employed in real-world environments. Conversely, in PVG, we can address real-world environments where supervision is not available since the action space is learned in an unsupervised manner As an alternative, the annotation effort required to address real-world videos can be reduced using a single action label to condition the whole video [140], but it limits interactivity since the user cannot control the generated action at each frame. Conversely, in PVG, the user can control the generation process by providing an action at every time-step after observing the last generated frame to achieve a greater degree of control.

This paper addresses these limitations introducing a novel framework for PVG named *Clustering for Action Decomposition and DiscoverY* (CADDY). Our approach discovers a set of distinct actions after watching multiple videos through a clustering procedure blended with the generation process that, at inference time, outputs playable videos. We adopt an encoder-decoder architecture where a discrete bottleneck layer is employed to obtain a discrete representation of the transitions between consecutive frames. A reconstruction loss is used as main driving loss jointly with an action loss based on a mutual information clustering procedure that avoid the need for neither action label supervision nor even the precise number of actions. A major difficulty in PVG is that discrete action labels cannot capture the stochasticity typical of real-world videos. To address this difficulty, we introduce an action network that estimates the action label posterior distribution by decomposing actions in a discrete label and a continuous component. While the discrete action label captures the main semantics of the performed action, the continuous component captures how the action is performed, such as the exact direction or speed of movement. At test time, the user provides only the discrete actions to interact with the generation process or optionally uses the continuous actions to control at a finer level the action to be executed such as speed of movement.

Finally, we experimentally demonstrate that our approach can learn consistent actions in varied environments. We conduct experiments on three different datasets including both real-world (*i.e.* tennis and robotics [24]) and synthetic (*i.e.* video game [7]). Our experiments show that CADDY generates high-quality videos while offering the user a better playability experience than existing future frame prediction methods.

In summary, our work brings the following contributions:

- **The problem of Playable Video Generation (PVG)** consisting in the unsupervised learning of video generators conditioned at each timestep on a set of discrete actions.
- **The CADDY framework for (PVG)**, consisting in an encoder-decoder network conditioned on a novel action representation consisting of a high-level discrete action component and a low-level continuous action component.
- **An action learning loss** based on a mutual information clustering procedure for the unsupervised discovery of discrete actions in input videos.
- **Two datasets** to foster future research on PVG, consisting in a real-world dataset of tennis matches and a synthetic Atari Breakout dataset.

2.2 Related Works

2.2.1 Video generation

Recent advances in deep generative models have led to impressive progress in video generation. A variety of formulations have been explored including Generative Adversarial Networks (GANs) [115, 135, 139], variational auto-encoders (VAEs) [4] and auto-regressive models [142]. While early works addressed the unconditional case [115, 139], many conditional video generation tasks have been addressed. The generated video can be conditioned on specific type of information. Among numerous examples, we can cite video to video translation [141], image animation [120, 121] or pose-based generation [10].

Another typical example is the problem of future frame prediction that consists in generating a video conditioned on the first video frames. Early works employ deterministic predictive models [25, 83, 139] that cannot handle the stochasticity inherent to real-world videos. Several methods have been proposed to incorporate stochasticity in the model using VAE formulations [29, 70, 135], GANs [68], or normalizing flows [65]. Inspired by VAE-based approaches, we adopt a probabilistic formulation that is compatible with our encoder-decoder pipeline. Nevertheless, our playable video generation framework goes beyond video prediction methods and allows the user to control generation of future frames with discrete actions.

To offer better control over the generation process than video prediction methods, some approaches condition the generated video on an input action label [58, 140]. However, these approaches require action annotations for training and the generated video cannot be controlled on-the-fly since a single label is used to generate the whole video. To further increase the level of control, other approaches condition each frame in the generated video on a different action label [16, 57, 97, 98]. In particular, Kim *et al.* recently proposed *GameGAN* [57], a framework that visually imitates a desired game. During training, GameGAN ingests a large collection of videos and the corresponding keyboard actions pressed by the player. The network is trained to predict the next frame from past frames and keyboard actions. While these methods allow playable video generation similarly to our approach, the requirement for frame-level annotation limits their application to constrained environments such as video games. Conversely, our approach does not use action label supervision and can be employed in real-world environments. In the context of future frame predictions, Rybkin *et al.* [113] propose to infer latent actions. However, this method learns a continuous action space that is later mapped to discrete actions with action supervision.

Few works have focused on the generation of videos with controllable characters in real-world environments [33, 151]. However, these approaches heavily rely on prior knowledge specific to the environment, such as pre-trained full-body pose estimators. On the contrary, in PVG, the goal is to design a general framework that can be applied to varied environments without modifications.

2.2.2 Deep clustering and unsupervised learning

Unsupervised learning has attracted growing attention with the raise of self-supervised learning techniques. Recent works address jointly clustering and representation learning so that the network is trained to categorize the training data while learning deep representation. For instance, DeepCluster [9] alternates between k-mean clustering and a feature learning phase where cluster assignments are used as supervision. As an alternative to this iterative algorithm, other approaches [51, 84] propose the use of networks that directly output cluster assignment rather than feature representations. In these methods, the network is trained with an information-theoretic criterion formulated as a mutual information maximization problem. Differently from these approaches that focus only on image clustering, CADDY tackles the problem of learning actions in videos by blending clustering with a generation formulation.

Note that, recent works address the problem of unsupervised learning for videos [23, 81, 102, 125] but they are either limited to representation learning [23, 81] or perform clustering at the video-level [102, 125] while PVG requires action labels at every time step.

2.3 Method

2.3.1 Overall Pipeline

In this work, we propose a framework where a user can interactively control the video generation process by selecting an action at every time step among a set of K discrete actions. Our method, named CADDY, is trained on a dataset of unannotated videos. We only assume that the video sequences depict a single agent acting in an environment. No action labels are required.

Inspired by Reinforcement Learning (RL) literature [130], the object in the scene is modeled as an agent interacting with its environment by performing an action at every time step. Differently from RL, our goal is to jointly learn the action space, the state representation that describes the state of the agent and its environment, and a decoder that reconstructs the observations (*i.e.* frames) from the state.

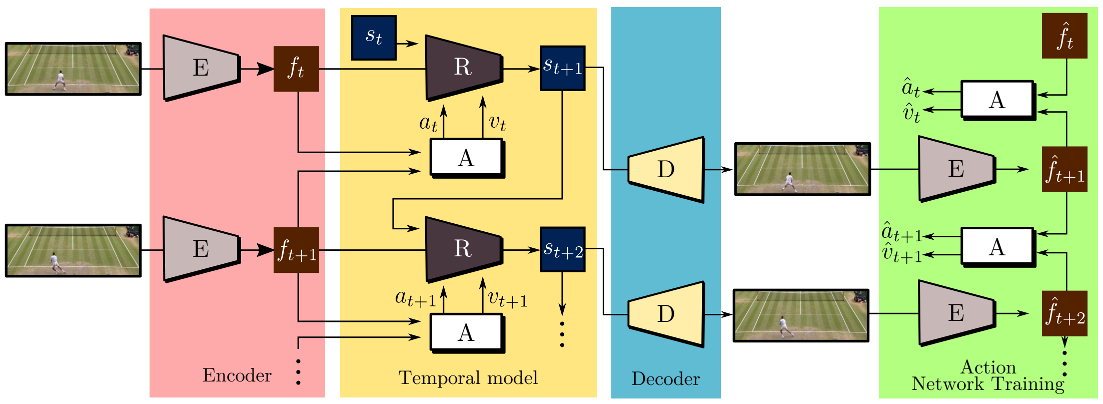


Figure 2.2: CADDY's training procedure for unsupervised playable video generation. An encoder E extracts frame representations from the input sequence. A temporal model estimates the successive states using a recurrent dynamics network R and an action network A which predicts the action label corresponding to the current action performed in the input sequence. Finally, a decoder D reconstructs the input frames. The model is trained using reconstruction as the main driving loss.

CADDY is articulated into four main modules illustrated in Fig. 2.2: (i) an encoder employs a network E to extract frame representations; (ii) a temporal model estimates the label corresponding to the action performed in the current frame and predicts a state s_{t+1} that describes the environment at the next time step, after performing the detected action. The action label is predicted via a network A that receives the frame representations from the current and next frames. To predict the next frame environment state s_{t+1}, we employ a recurrent neural network R that we refer to as *Dynamics network*. (iii) a decoder module employs a network D to reconstruct each frame from the frame embedding predicted by the temporal model. (iv) the reconstructed frames are fed to the encoder to assess the quality of the estimated action labels.

The overall pipeline is trained in an end-to-end fashion using as main driving loss a reconstruction loss on the output frames. The key idea of our approach is that the action network A needs to predict consistent action labels in order to correctly estimate the next frame embeddings s_{t+1}, and then, accurately reconstruct the input frames. We now describe each network.

Video encoder. Let us consider a video sequence $\{x_t\}_{t=1}^{T}$ of length T. First, for every frame, we extract a feature representation f_t with $t \in \{1, .., T\}$ which we call *frame features*. The aim of this representation is to encode information about the appearance and the semantics of the input frame:

$$f_t = E(x_t). \tag{2.1}$$

Action network. Next, the action network A is used to infer the discrete *action* $a_t \in \{1..K\}$ performed between the frames x_t and x_{t+1}. The action label a_t characterizes the transition between the features f_t and its successor f_{t+1}. In a deterministic environment, the evolution of the environment is fully described by a_t. However, in real scenarios this assumption is rarely satisfied and agents acting in a complex environment may express a behavior that can only be partially described with discrete actions. To handle this, we propose to model the transition between two consecutive frame features as the combination of the discrete action a_t and a continuous component v_t referred to as *action variability embedding*, *i.e.*:

$$a_t, v_t = A(f_t, f_{t+1}). \tag{2.2}$$

While the discrete action a_t describes the action performed by the agent at a high level, the continuous component v_t describes the variability of each action and captures the possible non-determinism in the environment.

In order to handle this non-determinism, we propose a probabilistic auto-encoder formulation to train the action network. In this regard, details are given in Sec. 2.3.2.

Dynamics network. Given the features f_t, the action a_t, and the corresponding *action variability embedding* v_t, the dynamics network R is used to embed the dynamics of the environment observed in the input sequence. We model the dynamics network as a recurrent network based on convolutional LSTMs [119] with the *environment state* s_t:

$$s_{t+1} = R(s_t, f_t, a_t, v_t). \tag{2.3}$$

The induction in Eq. (2.3) is initialized with a parameter s_0 that we consider to be trainable. The action label a_t and the *action variability embedding* v_t are concatenated channel-wise with the input feature f_t in the dynamics network.

Decoder. We use a decoder network D to reconstruct the frames $\hat{x}_{t+1}$ from the *environment state* s_{t+1}.

2.3.2 Probabilistic Action Network

As introduced earlier, the goal of the action network is to estimate the action label a_t and *action variability embedding* v_t. The action network first uses an *action state network* A_s to extract an *action embedding* e_t which represents the information in f_t that is related to the action currently performed. Following a probabilistic formulation, A_s predicts the posterior distribution of the *action embedding* modeled as Gaussian:

$$e_t \sim \mathcal{N}(\mu_{e_t}, \sigma^2_{e_t}) \tag{2.4}$$
$$\text{with } \mu_{e_t}, \sigma^2_{e_t} = A_s(f_t). \tag{2.5}$$

Similarly, the distribution of the *action embedding* $e_{t+1}{\sim}\mathcal{N}(\mu_{e_{t+1}}, \sigma^2_{e_{t+1}})$ is estimated from f_{t+1}. We can then combine the action states e_t and e_{t+1} to predict the performed action. More precisely, we propose to model the posterior distribution of the action (a_t, v_t) using the difference between the two *action embeddings* which still follows a Gaussian distribution. From the independence between random variables e_t and e_{t+1} we have:

$$d_t = e_{t+1} - e_t \sim \mathcal{N}(\mu_{d_t}, \sigma^2_{d_t}) \tag{2.6}$$
$$\text{with } \mu_{d_t} = \mu_{e_{t+1}} - \mu_{e_t} \text{ and } \sigma^2_{d_t} = \sigma^2_{e_{t+1}} {+} \sigma^2_{e_t}. \tag{2.7}$$

Finally, the *action direction* d_t is sampled according to (2.6) using the re-parametrization trick [60] and fed to a single-layer classifier to estimate the probability of each action $p_t{=}\{p^k_t\}^K_{k=1}{\in}[0,1]^K$. Importantly, we implement the classification layer using a Gumbel-Softmax layer [50] to obtain the discrete label a_t from the probabilities p_t while preserving differentiability for Stochastic Gradient Descent (SGD).

Regarding the *action variability embedding* v_t, considering $v_t{=}d_t$ would not favour the model to learn meaningful action labels a_t since the changes in the environment could be directly encoded in d_t without using a_t. Thus, we propose to make v_t dependent on both d_t and a_t in a way that d_t cannot be recovered from v_t alone, enforcing the learning of action labels a_t. To this aim, we consider a set of K *action direction centroids* $\{c_k\}^K_{k=1}$ (one per action) that are defined as the expected *action directions* for each action. Practically, the cluster centroids are estimated using an exponential moving average over the action direction associated with each discrete action. We propose to define the *action variability embedding* v_t as the difference between the observed action direction d_t and its assigned cluster centroid. Following a soft-assignement formulation, the *action variability embedding* is given by the expected difference with its action centroid:

$$v_t = \sum_{k=1}^{K} p_t^k (d_t - c_k). \tag{2.8}$$

In this way, v_t acts as a bottleneck for d_t since, assuming distinct centroids c_k, d_t cannot be completely encoded in v_t. To ensure distinct centroids, we introduce a loss that prevents their collapse to a single point (see Sec. 2.3.3).

2.3.3 Training objectives and test

CADDY is trained in an end-to-end fashion with a combination of objectives that aim at obtaining both high-quality output sequences and a discrete action space that captures the agent's high-level actions.

Reconstruction losses. The main driving loss of our system is a frame reconstruction loss based on the perceptual loss of Johnson *et al.* [53]:

$$\mathcal{L}_{\text{rec}}^{x} = \frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{J} \left\| N_j(x_t) - N_j(\hat{x}_t) \right\|_1 , \tag{2.9}$$

where N_j, $j \in \{1, ..., J\}$ indicates the j^{th} channel extracted from a specific VGG-19 layer and J is the number of feature channels in this layer. The reconstruction loss is averaged over different layers and resolutions [121]: D is equipped with multiple heads to output $\hat{x}_t$ at different resolutions and x_t is down-sampled to form a pyramid. This loss is completed by a L_1 reconstruction loss (denoted as $\mathcal{L}_1$) between $\hat{x}_t$ and x_t to improve convergence.

Additionally, we propose to assess reconstruction quality at the frame feature and action levels. The computation of these additional losses is represented in the green block of Fig. 2.2. The reconstructed frame $\hat{x}_t$ is fed to E leading to frame features $\hat{f}_t$. Then, the Euclidean distance between initial and reconstructed features is employed:

$$\mathcal{L}_{\text{rec}}^{f} = \frac{1}{T} \sum_{t=1}^{T} \left\| f_t - \hat{f}_t \right\|_2^2 . \tag{2.10}$$

The motivation for this loss is that it enforces that the semantic information extracted from input frames x_t is also preserved in $\hat{x}_t$. To avoid trivial minimization of (2.10), we do not back-propagate the loss gradient through f_t.

Action losses. Regarding action understanding, we propose an information-theoretic objective that acts on the action probabilities. The reconstructed *frame features* $\hat{f}_t$ and $\hat{f}_{t+1}$ are fed to A that internally estimates the action probabilities $\hat{p}_t$. We then maximize the mutual information between the actions extracted from the input sequences and the corresponding ones extracted from the reconstructed sequences:

$$\max_{\theta} \mathcal{MI}(p_t, \hat{p}_t) = \max_{\theta} (\mathcal{H}(p_t) - \mathcal{H}(p_t | \hat{p}_t)). \tag{2.11}$$

Maximizing this objective function has two desirable effects. First, minimizing the conditional entropy term $\mathcal{H}(p_t|\hat{p}_t)$ imposes that the same action must be inferred from the input and the reconstructed sequence. Second, maximizing the entropy term $\mathcal{H}(p_t)$ enforces that the maximal number of actions must be discovered. This avoids trivial solutions where only a single action label a_t is constantly predicted and ensures that the K *action direction centroids* $\{c_k\}_{k=1}^{K}$ do not collapse to a single point.

To compute the mutual information in Eq. (2.11), we consider a mini-batch $\mathcal{B}$ of action probabilities $(p_t, \hat{p}_t)$. We estimate the joint probability matrix $P_{ij} = P(a_t = i, \hat{a}_t = j)$, with $(i, j) \in \{1, ..., K\}^2$, as follows:

$$P_{ij} = P(a_t = i, \hat{a}_t = j) = \frac{1}{\bar{\mathcal{B}}} \sum_{(p_t, \hat{p}_t) \in \mathcal{B}} p_t . \hat{p}_t^\top, \tag{2.12}$$

where $\bar{\mathcal{B}}$ denotes the size of $\mathcal{B}$. We obtain $P_i = P(a_t = i)$ and $P_j = P(\hat{a}_t = j)$ by marginalization over P_{ij}. The mutual information loss term is then given by:

$$\mathcal{L}_{\text{act}} = -\mathcal{MI}(p_t, \hat{p}_t) = \sum_{t=1}^{K} \sum_{j=1}^{K} P_{ij} \ln \frac{P_{ij}}{P_i . P_j}. \tag{2.13}$$

This action matching loss $\mathcal{L}_{\text{act}}$ is completed by a loss that enforces that the *action variability embeddings* match. Considering the *action direction* distributions $d_t \sim \mathcal{N}(\mu_{d_t} \sigma^2_{d_t})$ and $\hat{d}_t \sim \mathcal{N}(\mu_{\hat{d}_t}, \sigma^2_{\hat{d}_t})$ associated with each $(p_t, \hat{p}_t)$ in $\mathcal{B}$, we measure the average Kullback–Leibler (*KL*) divergence between the distribution of $\hat{d}_t$ and d_t:

$$\mathcal{L}^a_{\text{rec}} = \frac{1}{\bar{\mathcal{B}}} \sum_{(\hat{d}_t, d_t) \in \mathcal{B}} \mathcal{D}_{\text{KL}}(\mathcal{N}(\mu_{\hat{d}_t}, \sigma^2_{\hat{d}_t}) \| \mathcal{N}(\mu_{d_t}, \sigma^2_{d_t})). \tag{2.14}$$

Assuming a Gaussian prior for d_t, we minimize the average *KL* divergence $\mathcal{D}_{\text{KL}}$ between the distributions of d_t and a Gaussian prior with unit variance $\mathcal{N}(0, I_n)$,

$$\mathcal{L}_{\text{KL}} = \frac{1}{\bar{\mathcal{B}}} \sum_{d_t \in \mathcal{B}} \mathcal{D}_{\text{KL}}(\mathcal{N}(\mu_{d_t}, \sigma^2_{d_t}) \| \mathcal{N}(0, I_n)). \tag{2.15}$$

Total loss. The total objective is given by:

$$\begin{aligned} \mathcal{L} = &\mathcal{L}_1 + \mathcal{L}^x_{\text{rec}} + \lambda^f_{\text{rec}} \mathcal{L}^f_{\text{rec}} + \lambda^a_{\text{rec}} \mathcal{L}^a_{\text{rec}} \\ &+ \lambda_{\text{act}} \mathcal{L}_{\text{act}} + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}, \end{aligned} \tag{2.16}$$

where $\lambda_{\text{act}}, \lambda_{\text{rep}}, \lambda_{\text{KL}}, \lambda^a_{\text{rec}}$ are positive weighting parameters. These parameters are estimated on the training set: for every term, we iteratively experiment raising parameter values until we can observe that the first back-propagation steps lead to a decrease of the loss term.

Test time. At test time, CADDY receives as input the initial video frame and the user provides an action label at every time-step. We follow an auto-regressive approach where the estimated $\hat{x}_{t+1}$ is used as input at the next time step (instead of x_{t+1}). We no longer employ the action network A and use the action labels a_t provided by the user. Regarding the *action variability embedding*, we employ $v_t = 0$ that corresponds to the maximum of its posterior distribution.

Note that, this difference between training and test would affect the performance of the network that has never received generated images as inputs. To mitigate this problem, we propose a mixed training procedure. For the first T_f frames, we use the training procedure as described above where the *Dynamic Network* R receives in input the *frame features* f_t computed from original frames. Then, for $t > T_f$, R is fed with the *frame features* $\hat{f}_t$ computed from the reconstructed frames $\hat{x}_t$. By mixing original and reconstructed frames, we obtain a network that can handle both real and reconstructed frames, and consequently, is less prone to the shift issue typical of auto-regressive methods.

2.4 Experiments

Datasets. We evaluate our method on three video datasets:

- *BAIR* robot pushing dataset [24]. We employ a version of the dataset in 256x256 resolution, composed of about 44K videos of 30 frames. Ground-truth robotic arm positions are available but are only used for evaluation purposes.
- *Atari Breakout* dataset. We collect a dataset using a Rainbow DQN agent [39] trained on the Atari Breakout video game environment. We collect 1407 sequences of about 32 frames with resolution 160x210 (358 for training, 546 sequences for validation and 503 for testing).
- *Tennis* dataset. We collect Youtube videos corresponding to two tennis matches from which we extract about 900 videos with resolution 256x96. To respect the single agent assumption, we consider only the lower part of the field.

Evaluation Protocol. We propose to evaluate both action and video generation qualities by comparing the models on the video reconstruction task. A test sequence is considered and the action network is used to extract the sequence of learned discrete actions characterizing the input sequence. Starting from the initial frame, the extracted actions are used to reconstruct the remaining of the sequence. The evaluation is completed with a user-study that directly assesses the quality of the learned set of discrete actions. We adopt a large set of metrics.

Video quality metrics.

- *LPIPS* [154]: We report the average *LPIPS* computed on corresponding frames of input and reconstructed sequences.
- *FID* [40]: We report the average *FID* between the original and reconstructed frames.
- *FVD* [136]: We compute the *FVD* between the original and reconstructed videos.

Action-space metrics. We introduce two metrics that measure the quality of the action space. Both metrics use additional knowledge (*i.e.* ground-truth information or an externally trained detector) to measure motion consistency among frames where the same action is performed. Assuming two consecutive frames, we measure the displacement Δ of a reference point on the object of interest. On the *Tennis* dataset, we employ FasterRCNN [109] to detect the player and use the bounding box center as the reference point. On *Atari Breakout*, we employ a simple pixel matching search to detect the rectangular platform and use its center as the reference point. Finally, on *BAIR*, we use the ground-truth location information of the robotic arm. The key idea of the two action quality metrics is to assess whether the predicted action labels and the displacement Δ are consistent by trying to predict one from the other:

- Δ *Mean Squared Error (Δ-MSE)*: This metric measures how Δ can be regressed from the action label. For each action, we estimate the average displacement Δ, which is the optimal estimator for Δ (in terms of MSE). We report the MSE to evaluate regression quality. To facilitate comparison among datasets, we normalize the MSE by the variance of Δ over the dataset.

	T_f	T_{initial}	T_{final}	*batch size*	K
Atari Breakout	6	7	9	8	3
BAIR	6	7	12	8	7
Tennis	6	7	12	6	7

Table 2.1: Hyperparameters used on the different datasets.

- *Δ-based Action Accuracy (Δ-Acc)*: This metric measures how the predicted action can be predicted from the displacement Δ. To this aim, we train a linear classifier and report the action-accuracy measured on the test set.

Action-conditioning metrics. Finally, we consider two metrics that measure how the action label conditions the generated video. Therefore, it evaluates both generation quality and the learned action space:

- *Average Detection Distance (ADD).* Similarly to [121], we report the Euclidean distance between the reference object keypoints in the original and reconstructed frames. We employ the same detector as for the action-quality metrics to estimate the reference points.

- *Missing Detection Rate (MDR)*: We report the percentage of detections that are successful in the input sequence, but not in the reconstructed one. This represents the proportion of frames where the object of interest is missing.

2.4.1 Implementation and Training details

Optimization. In all our experiments, we use an Adam optimizer with a fixed learning rate of $2e-4$.
Sequence length scheduling. During the initial phase of training, we notice greater stability with small values of T, while, on the other hand, training with large values of T increases the quality of long generated sequences. For this reason, we adopt a training scheme with a variable value of T. In particular, every 5000 iterations, we increase the current value of T by 1 until the target value. Tab. 2.1 shows the hyperparameter configurations for our datasets. In all the experiments, we set the initial value of T to 7 and use $T_f = 6$ context frames.
Estimation of the number of actions K. Our method requires the number of actions K to be provided as a hyper-parameter. For the *Atari Breakout* dataset, we pose $K = 3$ following the number of discrete actions in the real game. *BAIR* and *Tennis*, however, do not possess an intrinsic discrete action space, so the number of discrete actions required to capture the dynamics of the environment must be estimated. From initial experiments, we observe that this number can be over-estimated without negative consequences on the training process, with the model using extra actions to learn variations on the same action. On *BAIR*, we estimate $K = 7$ which allows 2 movements on each of the 3 axes to be learned, plus a no movement action. On *Tennis*, we also choose $K = 7$, expecting to learn 2 movements on the horizontal axis and 2 movements on the vertical axis, a *Stay* action, a *Hit the ball* action and an extra action that allows the model, if necessary, to learn an additional behavior of the player.
Gumbel-Softmax temperature annealing. Hard Gumbel-Softmax [50] discrete action sampling ensures that the action component a is truly discrete. Using a hard sampling strategy producing one hot vectors at the beginning of the training process, however, caused optimization difficulties. For this reason, we adopt a soft Gumbel-Softmax sampling approach. At the

beginning of training, we perform sampling with a temperature of 1.0 which does not enforce a very low entropy on the sampled action vector a. As the training progresses, we linearly reduce the sampling temperature to 0.4 at step 20.000, enforcing that the sampled values of a are similar to one hot vectors.

Loss weights. We follow the hyperparameter selection procedure explained in Sec. 3.3 of the main paper to estimate the loss weights on the *Tennis* dataset. The mutual information maximization loss λ_{act} is used with weight 0.15, λ_{rep} is set to 0.2, and λ_{KL} is used with weight $1e$–4, while λ^a_{rec} is posed to $1e$–5. We found that these same values produce similar optimization behaviors on the *BAIR* and on the *Atari Breakout* datasets, so we use the same loss weights for all the experiments.

Pretraining. We notice that convergence speed is increased if the encoder network E and the decoder network D are initialized to perform frame reconstruction. For this reason, we integrate a short pretraining phase into our approach. In particular, instead of computing s_t using the dynamics network R, we employ a small auxiliary network to directly translate f_t to s_t so that E and D can be trained in isolation on the reconstruction task. In this phase, the dynamics network produces a reconstruction of s_t which we call $\hat{s}_t$, and a reconstruction loss between s_t and $\hat{s}_t$ is imposed. Gradients for this loss, however, are propagated through $\hat{s}_t$ only. The other loss terms remain unaltered.

2.4.2 Experimental analysis of CADDY

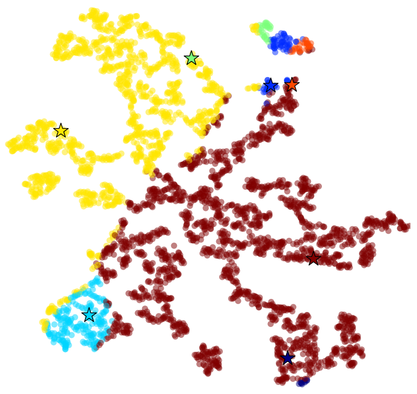
Ablation study. In this section, we study the impact of three key elements: Gumbel-Softmax sampling, the action loss $\mathcal{L}_{\text{act}}$ and *action variability embeddings* v_t. We produce the following variants of our method: (i) uses none of these components, (ii) introduces G.S., (iii) employs G.S. and $\mathcal{L}_{\text{act}}$, (iv) uses G.S. and v_t. The *BAIR* dataset is used because ground-truth displacement Δ are available.

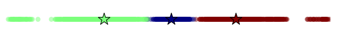
The results are presented in Tab. 2.2. In (i), when no component is used, the lack of G.S. sampling makes the model exploit p_t at training time to encode information about the next frame, learning a continuous rather than a discrete action representation. At test time, when discrete actions are used, this mismatch leads to poor performance. When G.S. sampling is introduced in (ii), the model learns discrete action representations with a Δ-*MSE* of 64.8%. However, the model lacks a mechanism to resolve the ambiguity of which variation of the discrete action should be performed. This results in difficulties in optimizing the reconstruction objective and leads to reduced video quality. In (iii), when both G.S. sampling and $\mathcal{L}_{\text{act}}$ are present, the optimization process favors $\mathcal{L}_{\text{act}}$ rather than the reconstruction objective, resulting in degraded performance. Lastly, in (iv), when both G.S. sampling and v_t are used without $\mathcal{L}_{\text{act}}$, the model uses v_t at training time to encode complete information about the next frame and a discrete action space is not learned. At test time, when v_t=0, performance of the model is poor. Overall, this ablation study confirms the positive impact of G.S sampling, the *action variability embeddings*, and our mutual-information loss on the performance.

Visualization. We analyze the learned space of *action directions* d_t. Fig. 2.3a shows the action space learned on *BAIR*, where CADDY discovers two predominant categories of actions that correspond to common small movements of the robot hand. The remaining action categories are assigned to less common actions including lifting or lowering the robot arm or making fast horizontal movements. Fig. 2.3b illustrates the learned action space for *Atari Breakout.* The model clearly divides the *action directions* into three clusters corresponding to left movement, no movement and right movement. In *Tennis*, as shown in Fig. 2.3c, the model learns a rich action space whose structure is correlated with player movement. According to AMT user votes, Action 6 (orange) corresponds to *Stay* and its corresponding *action directions* occupy the center of the space. Action 4 (green) and 7 (red) correspond respectively to right and left movement

Variant	G.S	v_t	$\mathcal{L}_{\text{act}}$	LPIPS↓	FID↓	FVD↓	Δ-*MSE*↓	Δ-*Acc*↑
(i)				0.263	80.0	1300	69.7	51.2
(ii)	✓			0.209	42.3	571	64.8	37.9
(iii)	✓		✓	0.249	76.4	1130	92.7	24.1
(iv)	✓	✓		0.245	76.9	1130	93.7	27.6
CADDY (Full)	✓	✓	✓	0.202	35.9	423	54.8	69.0

Table 2.2: Ablation results on the *BAIR* dataset. G.S: use of Gumbel-Softmax, v_t; use of the *action variability embedding* v_t; $\mathcal{L}_{\text{act}}$: training with the mutual information loss $\mathcal{L}_{\text{act}}$. Δ-*MSE* and Δ-*Acc* in %.


(a) *BAIR*


(b) *Atari Breakout*

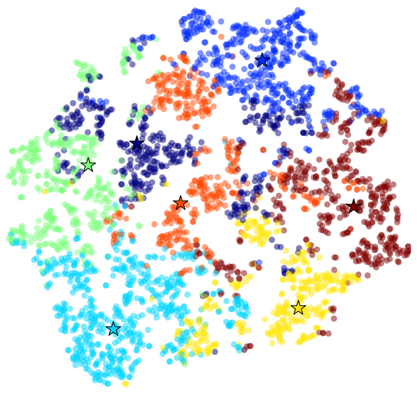

(c) *Tennis*

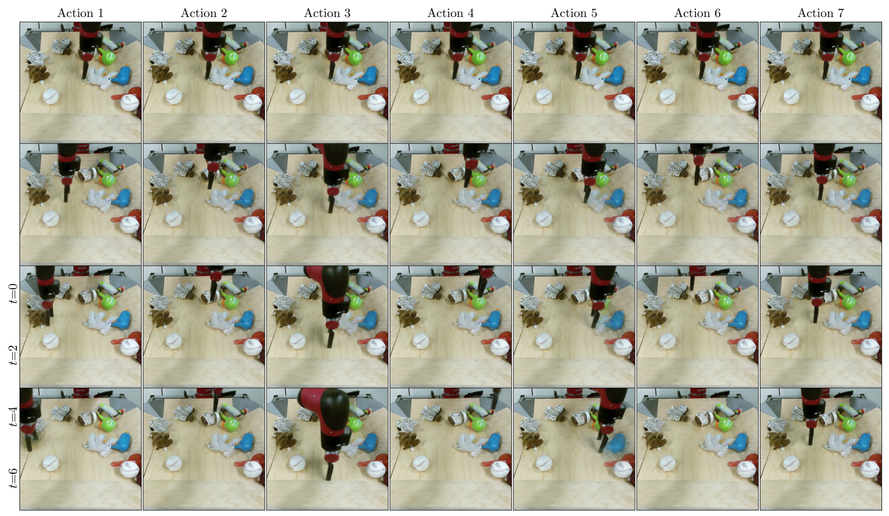
Figure 2.3: Visualizations of the learned space of *action directions* d_t with corresponding *action direction centroids* $\{c_k\}_{k=1}^{K}$, grouped by action.

and occupy opposing portions of the action space. Similarly, Action 2 (blue) and 3 (light blue) correspond to forward and backward movement and occupy opposing positions in the action space. Finally, according to human evaluation, Action 1 (dark blue) and Action 5 (yellow), which are positioned at boundary regions, have mixed correspondence to the movements of the neighboring regions and combine movement with ball hitting actions.

Fig. 2.4 shows the object motion distribution corresponding to each action learned by CADDY. On *BAIR*, the model learns actions related to movements on the x (1, 4), y (2, 3) and z (6) axes as well as a no-movement action (7). On *Atari Breakout*, the model learns actions corresponding to the three possible movements, as well as the inertia of the platform and the physics of the ball and blocks. On *Tennis*, the model learns actions corresponding to forward (2) and backward (3) movement, lateral movement (4, 7), no-movement (6) and hitting (1, 5).

In order to visualize in more detail the effects produced by each action learned by CADDY, we consider an initial frame and, for each action, we produce a sequence by repeatedly using the current action as user input. We show the obtained results in Fig. 2.5 and Fig. 2.6. On all the datasets, our model learns actions that correspond to movement of the object of interest along each axis. In addition, on the *Tennis* dataset, CADDY learns actions related to ball hitting.

Furthermore, we perform an evaluation of the capacity of *action variability embeddings* to capture variations of the relative action. In particular, we consider a pair of actions a_i and a_j. At inference time, since we pose *action variability embeddings* $v_i = v_j = 0$, for Eq. (8)



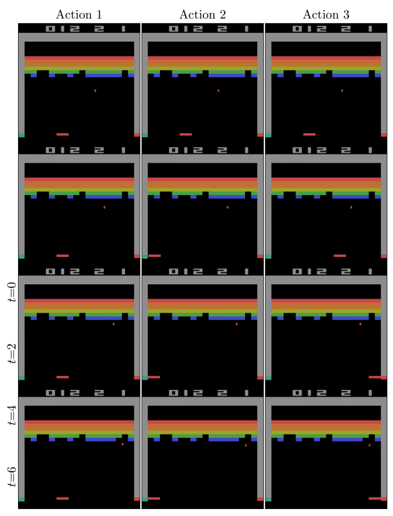


Figure 2.5: Videos generated by CADDY on the *BAIR* (left) and on the *Atari Breakout* (right) datasets. We generate a sequence for each learned action by repeatedly inputting the current action starting from the same initial frame. In all datasets, the model learns actions that correspond to movement on each axis.

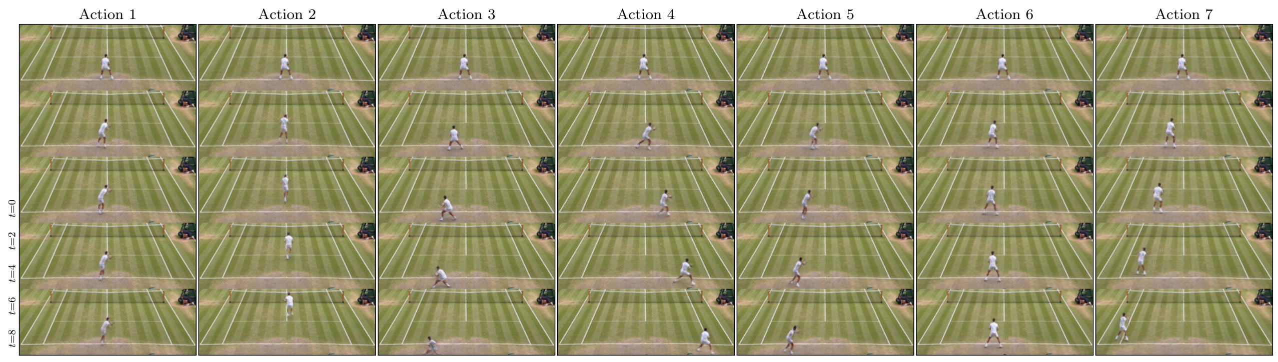
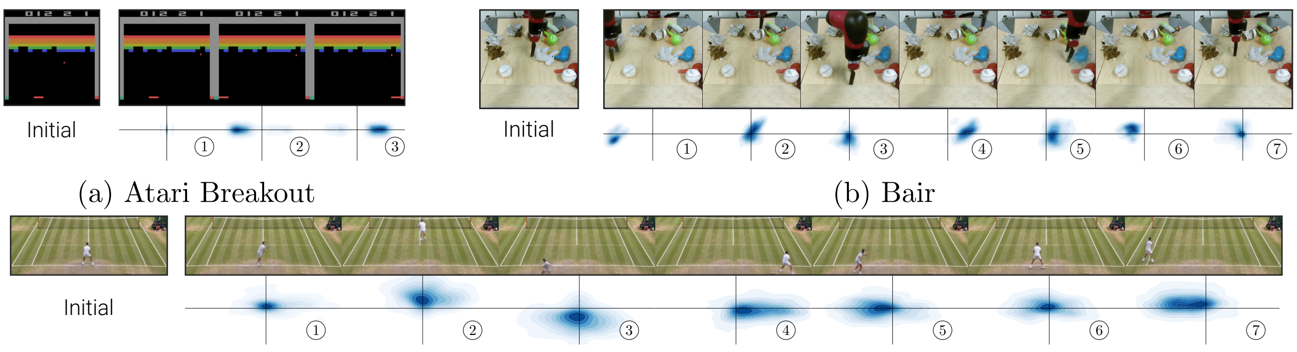


Figure 2.6: Videos generated by CADDY on the *Tennis* dataset. We generate a sequence for each learned action by repeatedly inputting the current action starting from the same initial frame. The model learns actions that correspond horizontal (Act. 4 and Act. 7) and vertical player movement (Act. 2 and Act. 3), no movement (Act. 6), and ball hitting (Act. 1, Act. 5).



(a) Atari Breakout (b) Bair

(c) Tennis

Figure 2.4: Qualitative evaluation of the action space learned by our model on the three datasets. We consider an initial frame and, for every action, produce a sequence by repeatedly selecting that action and show the final frame. The bottom row shows, for each action, the distribution of the displacement Δ associated with the object of interest.

we have $d_i = c_i$ and $d_j = c_j$ i.e. the associated *action directions* are centered on the *action direction centroids*. We argue that it is possible to produce actions with intermediate effects between a_i and a_j by sampling *action variability embeddings* corresponding to *action directions* in intermediate locations between the two *action direction centroids*. Let $l \in [0, 1]$ be an *interpolation factor*. We pose

$$a, c = \begin{cases} a_i, c_i & \text{if } l <= 0.5 \\ a_j, c_j & \text{if } l > 0.5 \end{cases}$$

$$v = l(c_j - c_i) + c_i - c$$

The resulting *action* a and *action variability embedding* v represent an intermediate action between a_i and a_j. In Fig. 2.7 we show qualitative results representing the effects obtained using intermediate actions that interpolate between a pair of discrete actions.

Lastly, we use CADDY to produce videos with direct user interaction. Starting from an initial frame, the user presses the button on its keyboard corresponding to the action to use at the current step, and the model generates the next frame. An extract of the generated results is shown in Fig. 2.8.

2.4.3 Comparison with Previous Methods

Baseline selection. Since we present the first method for unsupervised PVG, we compare the proposed model against a selection of baselines, adapting existing methods to this new settings. We consider two criteria in the selection of the baselines: the architecture should be adaptable to the new setting without deep modifications, and the source code must be available to include these modifications. Comparing existing video prediction methods in terms of FVD on a 64x64 version of the *BAIR* dataset shows that SAVP [70] is the best performing method with the code publicly available. Only the methods described in [17, 80, 142] are marginally outperforming SAVP but their code is not available.

Furthermore, SAVP [70] is widely used and features an architecture prone to be adapted to the current setting. During training, SAVP learns an encoder E that encodes information relative to the transition between successive frames which is used by the generator in the synthesis of the next frame. After training the SAVP model, we cluster the latent space learned by the encoder network on the training sequences using K-means. This procedure produces a set of K centroids that we use as action labels. During evaluation, we compute the latent

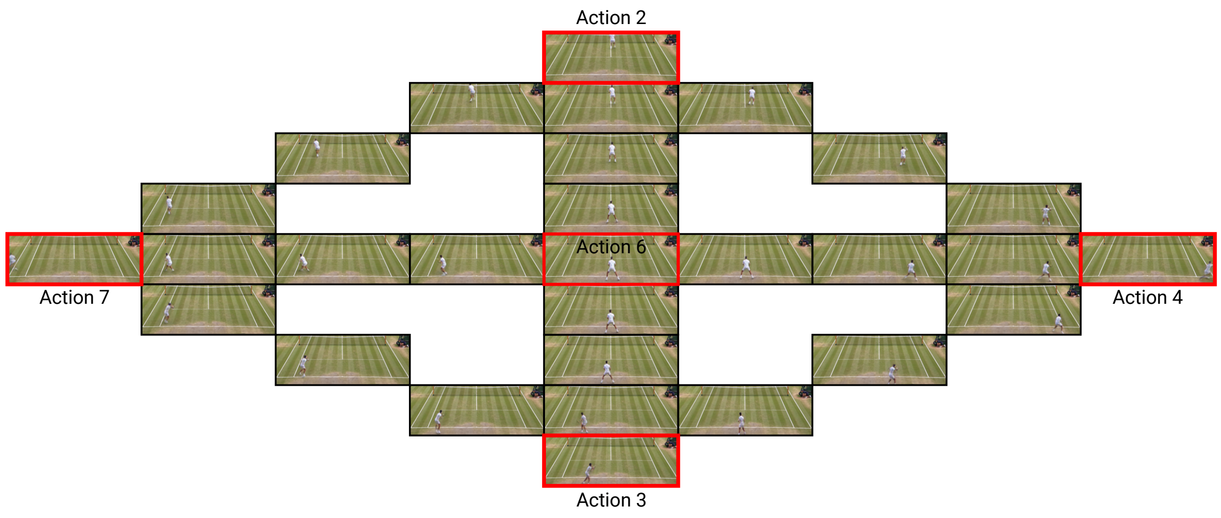


Figure 2.7: Visualization of the last frame of sequences produced from the same initial frame with differing *actions* and *action variability embeddings*. The sequences in red represent pure discrete actions where, *action variability embeddings* are posed to 0. For the sequences in black instead, *actions* and *action variability embeddings* are derived as described in Sec. 2.3.2 to represent intermediate actions between the two closest discrete actions using values of the *interpolation factor* of 0.3, 0.5 and 0.7. Note how the player positions depicted in the black sequences smoothly interpolate between the positions obtained with pure discrete actions, showing the capability of *action variability embeddings* to capture variations on discrete actions.

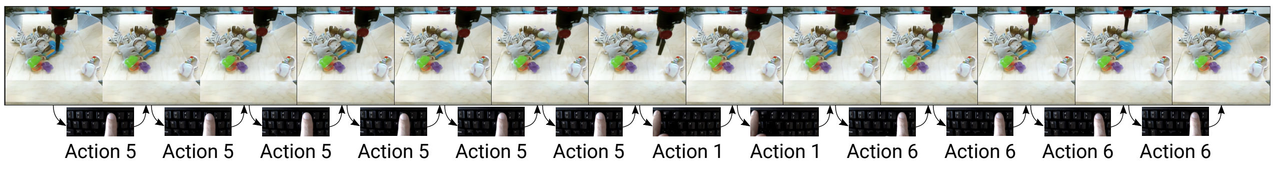
representations for the input sequence and assign them the index of the nearest centroid.

As a second baseline, we choose MoCoGAN [135]. This method possesses favorable characteristics for adaptation: it separates the content from the motion space and it employs an InfoGAN [15] loss to learn discrete video categories. We consider 6-frame short videos and assume a constant action over them. We employ an action discriminator that predicts the action performed in the input video sequence. This predicted action is used as an auxiliary input to the generator and the InfoGAN loss is used to learn the action space.

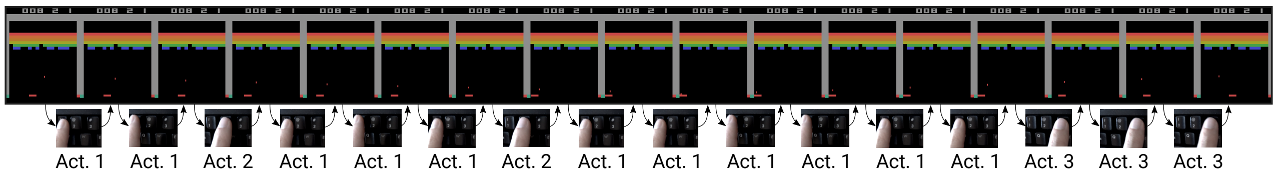
Finally, we also include SRVP [29] in our comparison. SRVP is modified similarly to SAVP [70] to handle PVG. However, due to the poor empirical results and the high training costs, we consider this baseline only on the *BAIR* dataset. Several other works have been considered, but not adopted [58, 145] since they would require too important modifications to handle the PVG problem.

SAVP [70], MoCoGAN [135] and SRVP [29] are designed for video generation at resolutions lower than our datasets. Therefore, we generate videos at the resolution of 64x80, 64x64 and 128x48 respectively for the *Atari Breakout*, *BAIR* and *Tennis* datasets, and then upscale to full resolution for evaluation. In addition, we create two other baselines, referred to as SAVP+ and MoCoGAN+, obtained by increasing the capacity of the respective networks to generate videos at full resolution. Due to the high memory requirements of SRVP [29], it was not feasible to produce a version operating at full resolution.

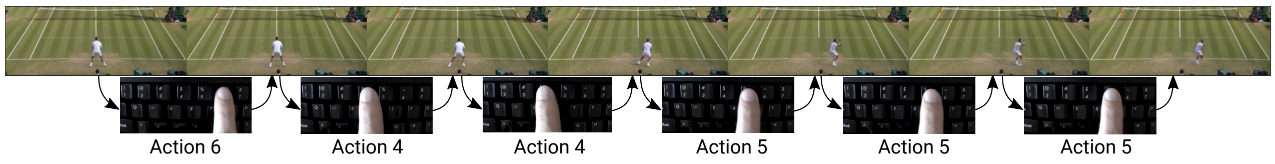
Quantitative evaluation. We show the results for the evaluation on the *BAIR* dataset in Tab. 2.3. According to Δ-*MSE*, MoCoGAN, SAVP and our model learn an action space correlated with the movement of the robot (Δ-*MSE*<100), but our method surpasses these baselines by 26.1% in Δ-*MSE* and by 24.2% in Δ-*Acc*, showing greater capacity in learning a set of discrete actions. Finally, SRVP clearly under-performs all the other methods in term of reconstruction (LPIPS), video quality (FID and FVD). Moreover, SRVP predicts a single action class for the whole test set. This degenerated behavior results in a 100% Δ-*ACC* and



(a) BAIR



(b) Atari Breakout



(c) Tennis

Figure 2.8: Video sequences generated by CADDY with direct user interaction on the *BAIR* (a), *Atari Breakout* (b) and *Tennis* (c) datasets.

a poor Δ-*MSE*. Because of these poor results and its very high computational requirement, SRVP is not shown in the following.

Tab. 2.3 reports the results on the *Atari Breakout* dataset. Our method outperforms the baselines in both action and video quality metrics. In particular, our model replicates the movement of the platform with an average ADD of 7.29 pixels and an MDR of 2.70%, indicating that the learned set of actions are consistent with the movement of the user-controlled *Atari Breakout* platform. Note that MoCoGAN obtains a lower MDR but a much higher ADD showing that it generates correctly the platform but at the wrong position.

The evaluation results on the *Tennis* dataset are reported in Tab. 2.3. With the exception of FVD, where the result is close to SAVP+, our method obtains the best performance in all the metrics. A Δ-*MSE* of 72.2% shows that actions with a consistent associated movement are learned. On the other hand, the Δ-*MSE* scores of the other baselines show that the movements associated with each action do not present a consistent meaning. Moreover, our method features the lowest ADD and a significantly lower MDR of 1.01% which indicate that our method consistently generates the player and moves it accurately on the field.

Qualitative evaluation of reconstruction quality In Fig. 2.9, we show examples of reconstructed sequences on the *BAIR* and *Tennis* datasets. Our method achieves a precise placement of the object of interest with respect to the input sequence.

User evaluation of action space. To complete our evaluation, we perform a user study on the *Tennis* dataset aimed at the evaluation of the learned action space. We sample 23 random frames that we use as initial frames. For each of them, we generate K continuations of the sequences, one for each action, each produced by repeatedly selecting the corresponding action. For each sequence, users are asked to select the performed action among a predefined set (*Left*, *Right*, *Forward*, *Backward*, *Hit the ball* or *Stay*). An additional option *Other* is provided in case the user cannot recognize any action. We measure user agreement using the Fleiss' kappa measure [26] that is commonly used to evaluate agreement between categorical ratings [27]. In addition, to validate that all the actions can be generated (*i.e.* detecting mode collapse), we compute the diversity of the action space, expressed as the entropy of the user-selected actions. Results are shown in Tab. 2.4. While all methods capture actions with high diversity, our

	BAIR				
	LPIPS ↓	FID ↓	FVD ↓	Δ-*MSE* ↓	Δ-*Acc* ↑
MoCoGAN [135]	0.466	198	1380	88.8	20.7
MoCoGAN+	0.201	66.1	849	98.4	22.9
SAVP [70]	0.433	220	1720	80.9	41.4
SAVP+	0.154	27.2	303	82.0	44.8
SRVP [29]	0.491	224	3540	(100)	(100)
CADDY (Ours)	0.202	35.9	423	54.8	69.0

	Atari Breakout					
	LPIPS↓	FID↓	FVD↓	Δ-*MSE*↓	Δ-*Acc*↑	(ADD, MDR)↓
MoCoGAN [135]	0.234	99.9	447	99.6	81.9	(46.0, 0.795)
MoCoGAN+	65.8e-3	10.4	103	103	57.5	(54.6, 17.4)
SAVP [70]	0.239	98.4	487	103	58.1	24.7 , 21.0
SAVP+	39.3e-3	4.84	84.4	104	85.6	15.8 , 51.5
CADDY (Ours)	7.66e-3	0.716	5.94	82.7	91.6	7.29 , 2.70

	Tennis					
	LPIPS↓	FID↓	FVD↓	Δ-*MSE*↓	Δ-*Acc*↑	(ADD, MDR)↓
MoCoGAN [135]	0.266	132	3400	101	26.4	28.5, 20.2
MoCoGAN+	0.166	56.8	1410	103	28.3	48.2, 27.0
SAVP [70]	0.245	156	3270	112	19.6	10.7 , 19.7
SAVP+	0.104	25.2	223	116	33.1	13.4 , 19.2
CADDY (Ours)	0.102	13.7	239	72.2	45.5	8.85 , 1.01

Table 2.3: Comparison with baselines on the *BAIR*, *Atari Breakout* and *Tennis* datasets. Δ-*MSE*, Δ-*Acc* and MDR in %, ADD in pixels. Parentheses indicate degenerated cases resulting in uninformative metrics (see details in the text).

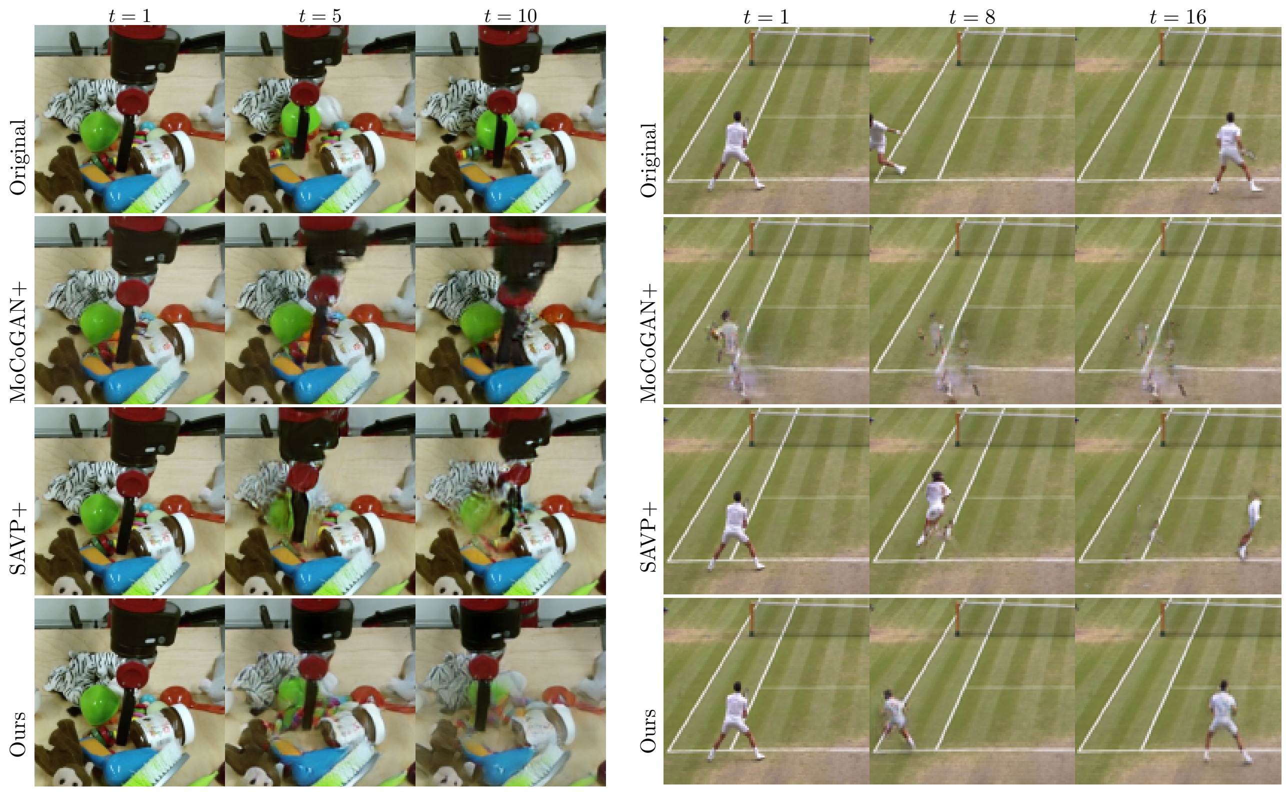


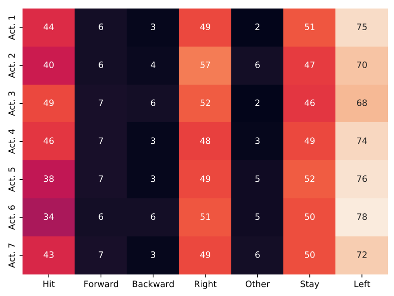
Figure 2.9: Reconstruction results on the *BAIR* (left) and *Tennis* (right) datasets. We zoom in for better visualization.

	Agreement	Diversity	*Other* votes
MoCoGAN [135]	-3.15e-3	1.58	1.80%
MoCoGAN+	-2.84e-3	1.51	28.0%
SAVP [70]	0.0718	1.69	7.14%
SAVP+	-1.97e-3	1.80	5.40%
CADDY (Ours)	0.469	1.65	1.61%

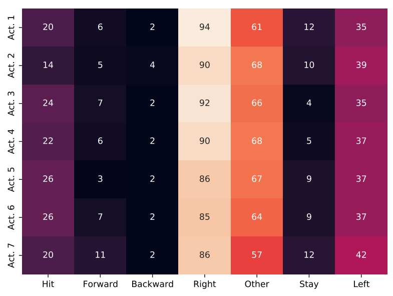
Table 2.4: User study results on the *Tennis* dataset.

method shows a higher agreement, indicating that users consistently associate the same action label to each learned action. In more detail, as shown in Fig. 2.10, the MoCoGAN, MoCoGAN+ and SAVP+ baselines do not learn a consistent action space. Indeed, their low Fleiss' kappa measures [26] show that users select different options for sequences generated with the same action, meaning that actions cause different effects based on the particular initial frame used to produce the sequence. On the other hand, the SAVP baseline (Fig. 2.10c) learns actions that result in a different distribution of user votes for each row, indicating a partial capability of the model to condition its output based on the input action. Differently from the other methods, CADDY (Fig. 2.10e) presents for each row a polarized response in a specific column, showing that our method associates the same meaning to an action independently from the starting frame. Some actions, including *Act. 1* and *Act. 5* present a lower user agreement. Despite the low number of votes given to the *Hit* action, a manual analysis of the corresponding sequences reveals that they correspond to the synthesis of ball hitting sequences, whose typical movement of the arm is difficult to spot and typically associated with movement of the player. This explains the portion of votes assigned to *Left*, *Right*, *Forward* and *Backward*.

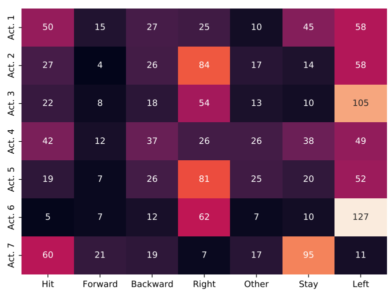
User evaluation of video quality. We perform an additional human evaluation on the *Tennis* dataset to assess the quality of the synthesized videos. Since our method produces



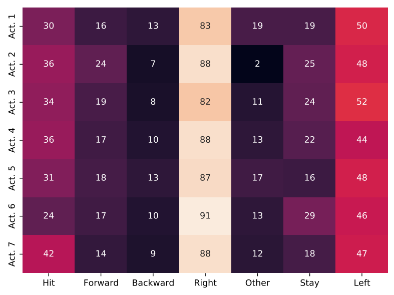
(a) MoCoGAN [135]



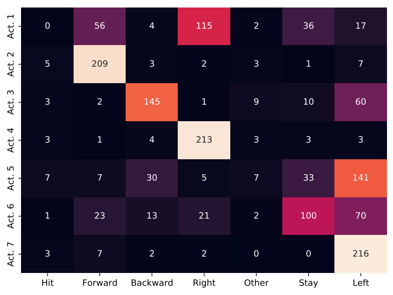
(b) MoCoGAN+



(c) SAVP [70]



(d) SAVP+



(e) CADDY

Figure 2.10: AMT votes for each method on the *Tennis* dataset. Rows correspond to the actions learned by the model, columns correspond to the action options presented to the users.

results in 256×96, we compare only with baselines producing outputs in the same resolution to ensure fairness, namely MoCoGAN+ and SAVP+. We run our study on AMT and ask users to express preference between one of two videos based on video quality. One video is produced with CADDY and the other with a baseline method. When compared to MoCoGAN+ and SAVP+, users express preference for our method in respectively 91.8% and 89.6% of cases.

2.5 Conclusions

In this chapter, we propose the unsupervised learning problem of playable video generation as a mean to allow the creation of video generators with increased degree of user control. We introduce CADDY, a self-supervised method based on an encoder-decoder architecture that uses predicted action labels as a bottleneck. To favor learning of actions, we introduce a decomposition of actions into a discrete high-level and a continuous low-level component that is learned through a combination of reconstruction and clustering-based action learning losses. We evaluate our method on three varied datasets and show state-of-the-art performance. Our experiments show that we can learn a rich set of actions that offer the user a gaming-like experience to control the generated video, demonstrating for the first time the possibility of building playable experiences based only on real-world videos.

While effective, the proposed framework present limitations in modeling videos with excessive camera motion and either allocates actions to explain such motion, or entangles agent actions with camera motion. In addition, the proposed framework is best suited for scenes with a single agent, as if multiple agents were present, the number of required actions to control all agents grows combinatorially, making them difficult to learn and use. In the next section, we introduce Playable Environments as a mean to address these issues.

Chapter 3

Playable Environments

In the preceding chapter, we presented a framework for the generation of playable videos operating on 2D sequences. Despite its strengths, this approach entails certain limitations, stemming from the implicit modeling assumption of solely utilizing a 2D representation. First, the model's ability to effectively disentangle agents from their background is hindered, as they are jointly modeled by the same 2D features, restricting its capacity to represent multiple agents simultaneously and to separate them from the background. In addition, it is challenging for the model to explain camera changes in the training sequences. Since no explicit camera model is provided, such movement can only be explained as actions, resulting in an undesired entanglement of camera movements with actions issued by the user at inference time. These limitations do not allow the framework to operate on more complex real-world video sequences with multiple agents taken from moving cameras. In conventional game modeling, 3D game engines are employed to represent scenes through compositions of 3D objects [35]. This acknowledgment of the world's inherent 3D nature facilitates seamless and independent manipulation of individual objects, allowing for their rendering from arbitrary camera perspectives. Inspired by this insight, in this chapter we propose the concept of "Playable Environments", a novel representation for interactive video generation and manipulation in both spatial and temporal domains. This new approach employs a composition of animatable 3D objects to represent the scene. With a single image at inference time, our novel framework allows the user to move objects in 3D while generating a video by providing a sequence of desired actions. The actions are learnt in an unsupervised manner. The explicit 3D nature of the framework enables the camera to be controlled to get the desired viewpoint. Our method builds an environment state for each frame, with fully represents the characteristics of each object such as their position, style an pose. The state of each object can independently be manipulated by our proposed action module and decoded back to the image space with volumetric rendering enabling a high degree of editability. To support diverse appearances of objects, we extend neural radiance fields with style-based modulation enabling applications such as environment or player identity swapping. Our method trains on a collection of various monocular videos requiring only the estimated camera parameters and 2D object locations which can be obtained through automatic procedures. To set a challenging benchmark, we introduce two large scale video datasets with significant camera movements. As evidenced by our experiments, playable environments enable several creative applications not attainable by prior video synthesis works, including playable 3D video generation, stylization and manipulation. All code and data are made publicly available.

3.1 Introduction

What would you change in the last tennis match you saw? The actions of the player? The style of the field, or, perhaps, the camera trajectory to observe a highlight more dramatically?

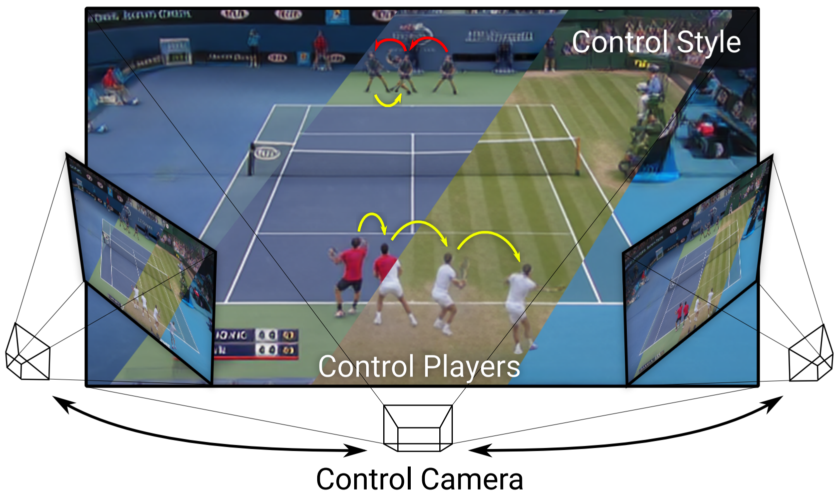


Figure 3.1: Given a single initial frame, our method creates *playable environments* that allow the user to interactively generate different videos by specifying discrete actions to control players, manipulating the camera trajectory and indicating the style for each object in the scene.

To do so interactively, the geometry and the style of the field and the players need to be reconstructed. Players' actions need to be understood and the outcomes of future actions anticipated. To enable these features one needs to reconstruct the observed *environment* in 3D and provide simple and intuitive interaction, offering an experience similar to *playing* a video game. We call these representations Playable Environments (PE).

Such a representation enables multiple creative applications, such as 3D- and action-aware video editing, camera trajectory manipulation, changing the action sequence, the agents and their styles, or continuing the video in time, beyond the observed footage. Fig. 3.1 shows a playable environment for tennis matches. In it, the user specifies actions to move the players, controls the viewpoint and changes the style of the players and the field. The environment can be played, akin to a video game, but with real objects.

In this chapter, we propose a method to construct PEs of complex scenes that supports a large set of interactive manipulations. Trained on a dataset of monocular videos, our method presents six core characteristics listed in Tab. 3.1 that enable the creation of such PEs. Our framework allows the user to interactively generate videos by providing discrete actions ⟨**1**⟩ and controlling the camera pose ⟨**2**⟩. Furthermore, it can represent environments with multiple objects ⟨**3**⟩ with varying poses ⟨**4**⟩ and appearances ⟨**5**⟩ and is robust to imprecise inputs ⟨**6**⟩. In particular, we do not require ground-truth camera intrinsics and extrinsincs, but assume they can be estimated for each frame. Neither do we assume ground-truth object locations, but rely on an off-the-shelf object detector [109] to locate the agents in 2D, such as both tennis players. No other supervision is required.

Playable Environments encapsulate and extend representations built by several prior image or video manipulation methods. Novel view synthesis and volumetric rendering methods support re-rendering of static scenes. However, while some methods support moving or articulated objects [99, 104, 134, 149], it is challenging for them to handle dynamic environments and they do not allow user interaction, making them undesirable for modeling compelling environments. Video synthesis methods manipulate videos by predicting future frames [66, 70, 133, 135], animating [120, 121, 122] or playing videos [85], but environments modeled with such methods typically lack camera control and multi-object support. Consequently, these methods limit interactivity as they do not take into account the 3D nature of the environment.

Our method consists of two components. The first one is the synthesis module. It extracts the state of the environment—location, style and non-rigid pose of each object—and renders

Name	Description
⟨**1**⟩ *Playability*	The user can control generation with discrete actions.
⟨**2**⟩ *Camera control*	The camera pose is explicitly controlled at test time.
⟨**3**⟩ *Multi-object*	Each object is explicitly modeled.
⟨**4**⟩ *Deformable objects*	The model handles deformable object such as human bodies
⟨**5**⟩ *Appearance changes*	The model handles objects whose appearance is not constant is the training set
⟨**6**⟩ *Robustness*	The model is robust to calibration and localization errors.

Table 3.1: Characteristics of our method for Playable Environments. Each row is referred in the text with ⟨·⟩ symbols.

the state back to the image space. Recently introduced Neural Radiance Fields (NeRFs) [89] represent an attractive tool for their ability to render novel views. In this chapter, we introduce a style-based modification of NeRF to support objects of different appearances. Furthermore, we propose a compositional non-rigid volumetric rendering approach handling the rigid parts of the scene and non-rigid objects. A major limitation of NeRF is its high computational complexity of the rendering operation. To address the issue and enable learning of high-resolution composable NeRF representations, we extend our NeRF to represent a field of features, and introduce an upsampler CNN operating on rendered feature maps. A perceptual loss component and patch-based training procedure are introduced to efficiently learn such representation. The second component—the action module—enables playability. It takes two consecutive states of the environment and predicts an action with respect to the camera orientation. We train our framework using reconstruction losses in the image space and the state space, and a novel loss for action consistency. Finally, to improve temporal dynamics, we introduce a temporal discriminator that operates on sequences of environment states.

To thoroughly evaluate ⟨**1**−**6**⟩, we introduce two complementary large-scale datasets for the training of playable environments, a synthetic and a real one. The first is intended to evaluate ⟨**1**−**5**⟩, with a particular focus on camera control thanks to the synthetic ground truth, the second to evaluate ⟨**1**−**6**⟩, with a particular focus on ⟨**4**−**6**⟩ given the high diversity present in this dataset. We propose an extensive evaluation of our method with several baselines derived from existing NeRF and video generation methods. These experiments show that our method is able to generate high-quality videos and outperforms all baselines in terms of playability, camera control and video quality.

In summary, the primary contributions of this work are as follows:

- **A new framework** for the creation of compelling Playable Environments with the characteristics in Tab. 3.1. No other concurrent framework enables generation and manipulation of video sequences to this degree.
- **A new compositional NeRF** that handles deformable objects with different visual styles. Our representation supports high-resolution videos of scenes with multiple agents thanks to a novel upsampler with patch-based training procedure.
- **An action module** that operates in the latent space of our NeRF model trained with an adversarial procedure to improve temporal temporal dynamics.
- **Two challenging large-scale datasets** for training and evaluating PEs to stimulate future research in this area.

3.2 Related Works

3.2.1 Video generation

Video generation has seen incredible progress over past years. The video synthesis task has numerous formulations which mostly differ in the type of conditional information that is used for generation. The generation process could be conditioned on previous frames [25,70,83,135, 139], on another video [120,121,122,141], on the pose of the agent [10] or even be completely unconditional [115,135]. Moreover, several works proposed to condition the generation of each single frame on an action label [16,57,97,98]. Still, all these methods require action supervision for training.

Playable video generation (PVG) was recently introduced in Menapace *et al.* [85]. Differently from prior works in this domain which required annotated action labels [55,57], their method, CADDY, automatically infers actions during training in a completely unsupervised manner from raw videos. This method is closely related to ours. However, CADDY assumes only a single controllable object while here we also model the camera movement, complex 3D interactions and support a variety of object appearances.

3.2.2 Novel view synthesis

Novel view synthesis methods traditionally utilized depth maps [12,103] or multi-view geometry [63,117,160] in order to reconstruct underlying 3D representation and later render new views of the corresponding scene. Recently, Neural Radiance Fields (NeRF) [89] revolutionized the field of novel view synthesis. The main idea of NeRF [89] is to model the scene as a continuous 5D function, usually represented by MLP, and directly query this function along the camera rays. Numerous follow-up works on [89] have been proposed. For instance, some works proposed to decompose the foreground and background [94,152]. Other works generalised NeRF [89] to dynamic scenes [52,99,134,149]. GIRAFFE [95] and GANcraft [37] proposed to utilize an internal representation that is rendered in a feature space and later decoded by a standard 2D convolutional network. However, none of these methods is able to generalize to multiple monocular videos, several moving and deforming objects, and diverse objects and scene appearances. Comparatively, our method can be trained with such data. Moreover, for enriching the interactivity of the playable environment, our method can control objects in the scene with action labels that are discovered in an unsupervised manner.

3.3 Method

Our framework is based on the encoder-decoder architecture shown in Fig. 3.2 whose design is driven by the playable environment characteristics ⟨**1-6**⟩ in Tab. 3.1. At time t, the encoder network outputs state vector s_t^i for every object i in the scene. To enable playability ⟨**1**⟩, we include an action module in the bottleneck layer that has two goals. First, it learns discrete action labels in an unsupervised manner. More precisely, we learn to discretize the transition from s_t^i to s_{t+1}^i using an action labels $a_t^i \in \{1, ..., K\}$, where the number of actions K is a hyper-parameter specified before training. Second, the action module is used at test time to condition the next frame generation on the action selected by the user. Finally, the decoder network, referred to as the synthesis module, is in charge of reconstructing the input frame combining the state of every object and the camera parameters to allow for camera control ⟨**2**⟩. The synthesis and action modules are trained in two separate phases using reconstruction as the main driving loss.

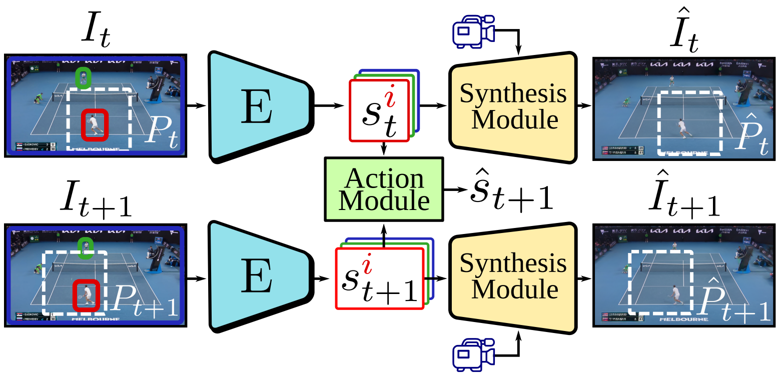


Figure 3.2: **Overview of our framework**. The encoder E extracts environment states for every object in the scene. The synthesis module follows a NeRF-like architecture to reconstruct the input frame and allows for camera manipulation. We introduce the action module that learns to encode state dynamics with discrete action labels. At test time, these learned action labels are provided by the user to control the generated content.

To handle environments with multiple objects ⟨**3**⟩, we adopt a compositional formulation for our encoder-decoder: we decompose the environment into a predefined set of objects. We distinguish between two object categories, namely static objects (*e.g.* background) and playable objects (*e.g.* human), where the latter are the dynamic objects the user will be able to control. We define the environment state of object i as $s_t^i = (x_t^i, w_t^i, \pi_t^i)$ where x_t^i is the position of the object in the environment, w^i is a style descriptor, and π^i is the object pose. We introduce w^i and π^i to handle deformable objects ⟨**4**⟩, such as humans, and to model appearance changes of objects in the training set ⟨**5**⟩. For every static object, we assume x_t^i to be fixed and known. For playable object i instead, given the current camera parameters and its bounding box b_t^i, we approximate x_t^i by projecting the middle point of the lower bounding box edge onto the ground plane. We then compute the style and pose descriptors using a convolutional encoder network E for each object. The encoder takes as input the image cropped at the location defined by the bounding box for each object and outputs both w_t^i and π_t^i. In the rest of the paper, we omit object indexes.

We introduce a novel synthesis module detailed in Sec. 3.3.1. The action module is described in Sec. 3.3.2. The training procedures are given in Secs. 3.3.3 and 3.3.4.

3.3.1 Synthesis Module

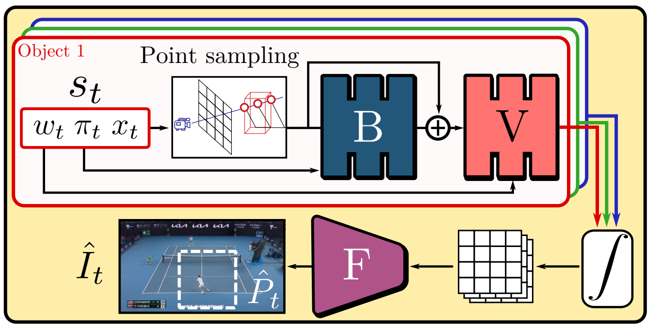


Figure 3.3: **The synthesis module** consists of two steps. First, non-rigid neural radiance fields with a bending network B and style modulation are used to generate a feature map. Second, the feature maps are fed to a ConvNet F.

The aim of the synthesis module is to reconstruct the input image from the camera pose and states s_t. We found NeRF [89] to be a reasonable base architecture for explicit camera control ⟨**2**⟩. Therefore, we propose a novel architecture (Fig. 3.3) that combines non-rigid neural radiance fields with a convolutional image generator to address ⟨**2**-**6**⟩.

Camera control ⟨**2**⟩ is achieved by employing NeRF [89] as a base architecture. NeRF [89] represents scenes as radiance fields: a 5D continuous function that maps a 3D position and a 2D viewing direction to an emitted color c and an opacity σ. Our NeRF represents scenes using a fully-connected network V, whose input is a single vector containing a point location in 3D. It outputs the volume density σ and radiance c for the input point location. Given a desired virtual camera, 3D points are sampled along the camera ray r traced through each pixel. The color value of every pixel is computed by integration over the ray r:

$$C(r) = \int_{t_n}^{t_f} e^{-\int_{t_n}^{t} \sigma(r(s))ds} \sigma(r(t))c(r(t))dt. \tag{3.1}$$

Similarly to [37, 95], instead of directly predicting color values, our neural radiance fields generate feature maps for the input camera pose, while a convolutional image generator is in charge of generating realistic frames. In addition, *Mildenhall et al.* [89] propose a stratified sampling approach which exploits a *coarse* neural radiance field to locate portions of the ray corresponding to visible surfaces and allocates additional samples to the neighborhood of such regions. A *fine* neural radiance field is used to compute the final image using both the initial and the additional sampled locations. We found stratified sampling not to improve performance of our method and thus use only a single neural radiance field with a uniform point sampling scheme.

For more details about NeRFs, please refer to [89].

Multi-object ⟨**3**⟩**.** Each object is modeled using a separate feature field parametrized as an object-specific MLP V. The field is bounded by volume β and centered at the respective object location x_t. Given a ray r, we compute its features $f(r)$ according to the following procedure. We first intersect r with each bounding volume β to compute the ingress and egress location of the ray with each object x_{in}, x_{out}. For each object, we then uniformly sample a given amount of positions $\{x_p\}_{p=1}^{N}$ between x_{in} and x_{out} and obtain the respective features f_p and opacities σ_p as $f_p, \sigma_p = V(x_p)$. $f(r)$ is obtained by integration similarly to Eq. (3.1).

Deformable objects ⟨**4**⟩**.** To handle deformable objects such as humans, we make use of non-rigid NeRF models, similarly to [134]. For each playable object, we introduce a ray bending network B parametrized as an MLP. Given an object pose descriptor π and position x_p on ray r, we use the bending network to regress the corresponding position $\tilde{x}_p$ on the bent ray $\tilde{r}$ as:

$$\tilde{x}_p = x_p + B(x_p, \pi_t). \tag{3.2}$$

We then make use of the positions on $\tilde{r}$ when sampling V. In this way, B encodes the transformation from the space of the deformed object to a canonical space and V encodes a canonical representation of the object.

Appearance changes ⟨**5**⟩**.** The appearance of each object may vary widely in the dataset. In order for each object-specific model to be able to represent the complete set of possible appearances of its object, we propose the use of a style embedding layer inspired by AdaIN [48], which we embed into V. Assuming a hidden feature h_t in V and a style code w_t, we modulate h_t as follows:

$$\tilde{h}_t = \gamma(w_t)h_t + \beta(w_t), \tag{3.3}$$

where γ and β are trainable linear layers. Following [89], we design V as a backbone terminated by two separate branches, one for opacity and one for features prediction. We assume that the style of an object should influence its features, but not its geometry. Therefore, we insert our modulation layer in the features prediction branch only.

Robustness ⟨**6**⟩ to calibration and localization errors is achieved through a Feature Renderer. Our compositional NeRF model outputs a feature map f_t corresponding to an input image

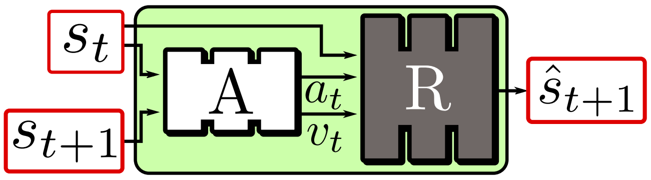


Figure 3.4: **The action module.** Given the states at times t and $t+1$, the action network A predicts a discrete action label a_t and action variability v_t that are combined by the dynamics network R to estimate the new environment state s_{t+1} given the old s_t.

patch. We employ a ConvNet F to reconstruct it. Due to the ability of ConvNets to model cross-pixel relationships, inaccuracies in the estimation of features caused by input noise can be compensated, reducing the associated blur. Note that F contains upsampling layers. It allows an important reduction in the number of rays that are to be sampled by the NeRF model since it outputs a feature map at a lower resolution than the image. Therefore, we reduce memory consumption allowing larger patches to be rendered. We also find it beneficial to use multiple input feature maps at different resolutions to capture details at different scales (see Sec. 3.4.1 for details).

3.3.2 Action Module

The action module (Fig. 3.4) learns the action space and enables playability $\langle\mathbf{1}\rangle$. The actions of each playable object are modeled by a separate action module, consisting of the action and dynamics networks.

Action network. Given two successive environment states s_t and s_{t+1}, we use an action network A to infer a discrete representation $a_t \in \{1, ..., K\}$ of the action performed by the object in the input sequence. Following [85], to address non determinism present in the environment, we also extract an action variability embedding v_t describing the particular variation of a_t performed at time t:

$$a_t, v_t = A(s_t, s_{t+1}). \tag{3.4}$$

Dynamics network. The role of the dynamics network is to predict the state s_{t+1} from s_t and the action label a_t. We adopt a recurrent model R implemented as an LSTM to model the dynamics of the object. The next state prediction $\hat{s}_{t+1} = (\hat{x}_t, \hat{w}_t, \hat{\pi}_t)$ is given by:

$$\hat{s}_{t+1} = R(s_t, a_t, v_t). \tag{3.5}$$

In our preliminary experiments, we observe that when R directly regresses $\hat{x}_{t+1}$ as formulated in (3.5), the model learns actions that are independent from the current camera position. This behavior is unnatural for the user since in applications such as video games, object movements are typically expressed relatively to the camera pose. To avoid this behavior, R is instead asked to predict the object movement Δ expressed in the camera coordinate system. The estimated position is then given by $\hat{x}_{t+1} = x_t + M\Delta$ where M is the rotation matrix expressing the orientation of the camera.

3.3.3 Synthesis Module Training

We train our model in two steps by first training the encoder and synthesis module until convergence, and then the action module. We train the encoder and synthesis module using the perceptual loss of Johnson *et al.* [53] that assesses image reconstruction quality in features spaces of a pretrained VGG network. The loss is computed between the ground truth and reconstructed image patches. The perceptual loss is complemented by an L2 reconstruction loss in the pixel space.

Our preliminary experiments showed that training may fail to correctly disentangle object style and pose (*i.e.* w and π respectively). Indeed, the reconstruction losses can be minimized using w alone by predicting a constant, non-deforming surface with changing style. To avoid this problem, we make the observation that the pose of an object in neighboring frames can change while the style does not. Therefore, we enforce better disentanglement by permuting the order of w codes along the temporal dimension for each sequence before feeding them to the synthesis module.

3.3.4 Action Module Training

In the second phase of training, we train the action module using a combination of losses. Each loss is computed separately for each playable object and then averaged to produce the final optimization objective.

Reconstruction loss. For each playable object, we reconstruct the input sequence of environment states $\{s_t\}_{t=1}^{T}$, obtained by encoding each input image using the encoder E, and impose an ℓ_2 reconstruction loss $\mathcal{L}_{\text{rec}}$ with the corresponding reconstructed sequence $\{\hat{s}_t\}_{t=1}^{T}$.

Action learning losses. We employ the information-theoretic action learning loss of [85] to foster the understanding of actions. For each playable object, the action network A produces internal estimates of action probabilities p_t and $\hat{p}_t$ for input s_t and reconstructed $\hat{s}_t$ environment states, respectively. By imposing the maximization of mutual information between these two distributions we foster the action network both to discover the K action categories, avoiding mode collapse, and to produce consistent action estimates for the input and reconstructed sequence:

$$\mathcal{L}_{\text{act}} = -\mathcal{MI}(p_t, \hat{p}_t). \tag{3.6}$$

In addition, to improve consistency between discrete actions and 3D movements, we propose to optimize a novel loss consisting in a soft version of the Δ Mean Squared Error (Δ-MSE) introduced in [85]. This metric is based on the idea that same actions a_t should correspond to similar object motions Δ. Assuming a batch containing J image pairs, we extract the object motion $\Delta_j, j \in \{1, ..., J\}$ and estimate the mean object motion for each action:

$$\forall k \in \{1, ..., K\}, \mu_k = \frac{\sum_{j=1}^{J} p_{jk} \Delta_j}{\sum_{j=1}^{J} p_{jk}}, \tag{3.7}$$

where p_{jk} denotes the probability for the image pair j to be assigned to the action k. We then minimize the mean squared distance between the motion Δ_j and the mean motion for each action:

$$\mathcal{L}_{\Delta} = \frac{1}{\text{Var}(\Delta)} \sum_{j=1}^{J} \sum_{k=1}^{K} p_{jk} \left\| \Delta_j - \mu_k \right\|_2^2, \tag{3.8}$$

where $\text{Var}(\Delta)$ is used as a normalization factor.

Temporal discriminator. Previous methods for playable video generation [85], tend to produce sequences where the playable objects move in the scene with unrealistic motions. We attribute this behavior to the use of reconstruction as the main training objective. Optimizing reconstruction losses does not penalize action representations that lead to temporally inconsistent videos. To address the problem, for each playable object we introduce a temporal discriminator D implemented as a 1D ConvNet over the temporal dimension. Given a sequence of environment states, the temporal discriminator is trained to classify them as real if produced by encoding the input images using E or as fake if reconstructed by the action module. We implement our adversarial training procedure using a vanilla GAN loss with loss terms $\mathcal{L}_{\text{G}}$ and $\mathcal{L}_{\text{D}}$ for the action module and temporal discriminator, respectively.

Total loss. Our optimization objective for A and R is

$$\mathcal{L} = \lambda_{\text{rec}} L_{\text{rec}} + \lambda_{\text{act}} \mathcal{L}_{\text{act}} + \lambda_{\Delta} \mathcal{L}_{\Delta} + \lambda_{\text{G}} \mathcal{L}_{\text{G}}, \tag{3.9}$$

where we introduce the weighting parameters λ_{rec}, λ_{act}, λ_{Δ} and λ_{G}. For training D, we minimize the adversarial objective $\mathcal{L}_{\text{D}}$ of the discriminator.
Inference. At inference time, we assume that only the first frame of the sequence is given. We use the encoder module to extract the first environment state $\hat{s}_1 = s_1$. At each timestep t, we let the user specify a discrete action for each playable object and use the dynamics network R to derive $\hat{s}_{t+1}$ in an autoregressive way. Since the action input is specified by the user, during inference we do not make use of the action network and always set $v_t = 0$. The environment states generated by the dynamics network are rendered to images using the synthesis module.

3.4 Experiments

Datasets. Evaluating ⟨**1-6**⟩ is challenging and requires video datasets featuring camera motion ⟨**2**⟩, multiple playable objects ⟨**1**,**3**⟩, deforming objects ⟨**4**⟩ and varied appearance ⟨**5**⟩. For this reason, we collect three datasets:

- *Minecraft* dataset. We collect a synthetic video dataset with duration of 1h with two sparring *Minecraft* [1] players. Wide camera movement and diverse, deforming players allow the evaluation of ⟨**1**−**5**⟩.
- *Minecraft Camera* dataset. We collect *Minecraft* [1] sequences where the camera is moved in the neighborhood of a starting position. We use these frames as a synthetic ground truth for the evaluation of camera control ⟨**2**⟩.
- *Tennis* dataset. We collect a large-scale dataset of 43 broadcast tennis matches totalling 12h of videos for the evaluation of ⟨**1-6**⟩. The dataset features challenging player poses ⟨**5**⟩, high variability in tennis fields and players ⟨**4**⟩ and noise in camera estimation and player localizaton ⟨**6**⟩.

To allow comparison with playable video generation methods under their simplifying assumptions, we adopt the *Tennis* dataset of [85], referred to as *Static Tennis*. The dataset features limited camera movement, each video is cropped to depict only a single player, only one field is present and players have uniform appearance, thus only ⟨**1**,**4**⟩ are evaluated.
Evaluation protocol. We perform a separate evaluation of the synthesis ⟨**2-6**⟩ and the action modules ⟨**1**⟩ using similar evaluation protocols. For the former, we reconstruct each test sequence by extracting the environment state of each frame and rendering the original frame back. For the action module, we follow the evaluation protocol of [85]. In particular, we consider a test sequence and extract the environment state of the first frame, then we use the action network to extract the sequence of discrete actions present in the sequence and reconstruct each frame starting from the first environment state. As video quality metrics ⟨**2**,**4-6**⟩ we adopt *LPIPS* [154], *FID* [40] and *FVD* [136] computed between the test sequences and the reconstructed sequences. For evaluation of the action space ⟨**1**,**3**⟩, following [85], we define Δ as the difference in position of an object between two given frames and use the following metrics:

- Δ *Mean Squared Error (Δ-MSE)*: The expected error in terms of MSE in the regression of Δ from a discrete action. For each action, the average Δ is used as the optimal estimator. The metric is normalized by the variance of Δ.
- Δ-*based Action Accuracy (Δ-Acc)*: The accuracy with which a discrete action can be regressed from Δ.

- *Average Detection Distance (ADD)*: The average Euclidean distance between the bounding box centers of corresponding objects in the test and reconstructed frames.
- *Missing Detection Rate (MDR)*: The portion of detections that are present in the test sequences but that are not matched by any detection in the reconstructed sequences.

3.4.1 Implementation and Training details

Architecture details. We make use of positional encodings [89] for both V and the bending network B. We use 10 and 6 octaves, respectively, and use the concatenation of the original vector and the encodings as input. Following [100], we gradually introduce the positional encodings in B during the first 60.000 training iterations. No viewing direction is given as input to B and V to avoid artifacts when the environment is rendered from a camera pose outside of the training distribution. In addition, to reduce memory consumption, we do not make use of the hierarchical point sampling scheme. We model the action network A following [85] and make use of gumbel softmax sampling [50] to obtain discrete action representations. In addition, to foster A in predicting actions that are associated to movement, we use only object positions x_t as input rather than the complete environment state s_t. The temporal discriminator D receives as input the object positions x and object poses π and is further conditioned on the actions a and action variabilities v inferred on the ground-truth sequence. We modulate the number of residual blocks in E based on the expected size of the input image crop.

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

$$m \in \left\{\frac{d_i}{2} + 1 + kd_i\right\}_{k=0}^{\frac{h}{d_i}-1}, n \in \left\{\frac{d_i}{2} + 1 + kd_i\right\}_{k=0}^{\frac{w}{d_i}-1}, \tag{3.10}$$

where h and w are the image height and width and indexes are expressed starting from 1. The process is illustrated in Fig. 3.5. Note that each ray is sampled on a grid where the points are at a vertical and horizontal distance of d_i from one another and at distance $d_i/2$ from the border (see left of Fig. 3.5 for a visualization of the sampled positions). Note also that the ray is sampled at the pixel location in the original image that corresponds to the center of the associated local patch that will be synthesized by F. In our implementation, we render two separate feature maps at downsampling factors 8x and 4x (see Fig. 3.5) to capture details at different scales. Note that this approach allows rendering images using 12.8x fewer rays than NeRF approaches that render each pixel with a separate ray, allowing for important reductions in the use of memory and in computational complexity and enables the model to render large image patches at training time. We experiment with greater downsampling factors, but find that further increasing them causes 3D consistency artifacts. In the presence of calibration and localization noise during training, we observe that NeRF models tend to generate blurry results which are particularly evident on dynamic objects where calibration and localization errors are compounded with errors in the estimation of the object pose. We ascribe this phenomenon to the use of reconstruction losses based on L2 distance which, in the presence of a color mismatch between the ground truth and the reconstructed pixel caused by input noise, favors the prediction of intermediate color values, generating blur. Using a feature renderer provides two advantages over the traditional approach. First, thanks to convolutional filters, the prediction for the feature associated to the current position can take into account the values of neighboring positions, enabling the model to produce a coherent output. Second, the possibility to render large patches makes it possible to adopt losses different from L2 distance,

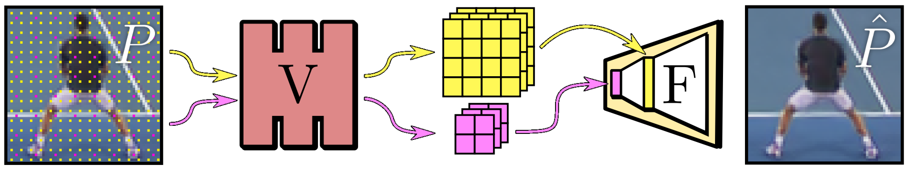


Figure 3.5: Representation of the feature rendering process. We sample points on rays arranged in a grid corresponding to feature map locations, then the sampled points are rendered into feature maps using our NeRF pipeline. The process is repeated to produce feature maps at different resolutions. The feature rendered F uses the rendered feature maps to reconstruct the original image.

such as the perceptual loss of *Johnson et al.* [53] which penalizes more effectively implausible predictions such as the ones containing blur.

Object configurations. On the *Minecraft* dataset, we model the scene using four objects. The first object models the static scene elements close to the players, the second object models the distant objects and the other two objects model the players. For each ray, we sample 16 positions for the first model, 1 position for the second model and 32 positions for each of the players. On the *Tennis* dataset, we note that camera movement in the input dataset is mostly limited to camera rotations. Recovering depth information for the static objects is thus an ill-posed problem which requires prior knowledge. Note that this problem is not present for the players since their change of position with respect to the camera allows the learning of depth information. We thus adopt the following objects to model the environment: an object to model the tennis field which is bounded by a box β that does not rise above ground level, an object to model the backplate of the tennis field, which is bounded by its box β to be a planar surface, and two objects for the players. For each ray, we sample 4 positions for the objects modeling the static scene and 32 positions for each player.

Background modeling. The *Minecraft* dataset features a challenging environment with distant visible objects. NeRF++ [152] employs an inverted sphere scene background representation for modeling distant objects. While effective, this parametrization requires a large number of samples to model distant objects, which increases memory consumption. To address this issue, we propose to model background objects using a spheric background representation which requires a single sample for each ray. In particular, we model the spheric background with as an MLP $f = V(d, o)$, receiving as input the direction d and the origin o of the ray, and returning as output the associated feature. We consider the opacity σ of the associated feature to always be 1. The parametrization on the ray origin allows the model to simulate the effects of depth on the background objects without requiring multiple samples.

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

Masked batch normalization and pooling. In a video sequence, detection may fail in some frames. In the action module, the environment state corresponding to frames with missing detections and the successive ones are replaced with placeholder values. As discussed in Sec. 3.3.4, loss masking is used to prevent effects of placeholder values on training. Nevertheless, the presence of placeholder values may still alter the behavior of the action module by affecting the estimation of batch statistics in batch normalization layers and by affecting the behavior of operations such as global average pooling. To prevent potential undesired effects on training, in the action module we adopt masked versions of batch normalization and of pooling operations that operate by computing statistics only on non-placeholder values.

Synthesis module training details. We optimize the synthesis module parameters using Adam [59] as optimizer and learning rate $5e-4$. The model is trained for 300.000 iterations and the learning rate is exponentially decayed until $5e-5$ at the end of training. We employ a batch size of 8 video sequences, each with 3 or 4 frames for *Minecraft* and *Tennis*, respectively. To better disentangle style from pose (see Sec. 3.3 on the main paper) we do not use consecutive frames for each sequence, but skip 4 and 100 frames between each selected frame, respectively, for *Minecraft* and *Tennis*. In all the experiments, we consider input images of size 512x288 and we render patches of size 192x192 and 256x256, respectively, for *Minecraft* and *Tennis* during training. To improve the quality of playable objects, we sample with increased frequency patches that contain the players. The total loss is the weighted sum of the perceptual and L2 reconstruction losses in pixel space. We assign a weight of 1.0 to the L2 reconstruction loss and a weight of 0.1 to the perceptual loss term. The synthesis module is trained on 4x Nvidia RTX 8000. We also train models on 1x Nvidia RTX 8000 on the *Tennis* dataset using a reduced rendered patch size of 160x160 pixels with a small reduction in image quality.

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

Action module training details. For the action module, we make use of Adam [59] as optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$, and use a learning rate of $5e-4$. We train the model for 300.000 iterations with an exponentially decayed learning rate which reaches the value of $5e-5$ at the end of training. The action module and the discriminator are optimized in alternation. We regularize the discriminator training using spectral normalization [90]. We note that at inference time our autoregressive dynamics network receives as input sequences of reconstructed environment states $\hat{s}_t$ rather than environment states s_t produced by E as happens during training. Following [85], to avoid performance degradation at inference time due to this mismatch, we propose to train the action module using as input to the dynamics network encoded environment states for the first t steps and reconstructed environment states for the following ones. In all the experiments we use 4 initial encoded environment states. In addition, we employ a curriculum learning strategy where we linearly increase the length of the reconstructed sequences during training. In particular, at the beginning of training, sequences of 5 environment states are reconstructed, while at the end of the annealing period at step 25.000 the reconstructed sequence length is set to 9. We use a batch size of 64 sequences. Following [85], we set the number of actions K to 7 for all the experiments. We set $\lambda_{\text{rec}} = 1.0$, $\lambda_{\text{act}} = 0.15$, $\lambda_{\Delta} = 0.1$ and $\lambda_{\text{G}} = 0.1$ in the computation of the total loss term. During training of the action module, some frames may not contain a valid detection for each playable object. This may be due to the absence of the object in the frame or due to missed detections. We address this issue by computing the loss terms $\mathcal{L}_{\text{rec}}$, $\mathcal{L}_{\text{act}}$ and $\mathcal{L}_{\Delta}$ by taking into account only the prefix of the input sequence where the object has always been detected. The action module is trained on 1x Nvidia RTX 8000.

	LPIPS↓	FID↓	FVD↓	Δ-*MSE*↓	Δ-*Acc*↑	ADD↓	MDR↓
MoCoGAN [135]	0.266	132	3400	101	26.4	28.5	20.2
MoCoGAN+	0.166	56.8	1410	103	28.3	48.2	27.0
SAVP [70]	0.245	156	3270	112	19.6	10.7	19.7
SAVP+	0.104	25.2	223	116	33.1	13.4	19.2
CADDY [85]	0.102	13.7	239	72.2	45.5	8.85	1.01
(Ours)	0.089	15.3	237	32.8	68.1	9.47	0.15

Table 3.2: Comparison with PVG state of the art on the *Static Tennis* dataset of [85]. Δ-*MSE*, Δ-*Acc* and MDR in %, ADD in pixels.

3.4.2 Comparison on Playable Video Generation

In this section, we evaluate the action-modeling capabilities of our method by comparing against the state of the art in the related problem of Playable Video Generation (PVG) [85] where the objective is to learn a set of discrete action labels in an unsupervised fashion to condition video generation. Differently from our setting, in PVG no explicit camera control is required. Moreover, existing PVG methods assume a single user-controllable object and that camera motion is limited.

To satisfy these simplifying assumptions, we adopt the *Static Tennis* dataset of [85]. Tab. 3.2 shows the results. Our method substantially improves the Δ-MSE and Δ-Acc action quality metrics suggesting that the learned actions are better correlated with player movement. In addition, the reduced LPIPS and MDR indicate an improvement in the quality of the generated reconstruction. In order to further evaluate video quality improvements with respect to [85], we run a user study on video quality against [85] on the *Static Tennis* dataset, the original dataset of [85]. Note that, for fairness of comparison, we make use of the *Static Tennis* dataset which does not feature the several challenges addressed by our method but not by CADDY [85], such as wide camera movements, multiple players and changes in appearance.

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

3.4.3 Comparison with Previous Methods

Baselines. We propose to build baselines for the creation of PEs from state-of-the-art methods in the related problem of PVG. We make use of the following set of versions of CADDY [85] which are modified to account for multiple playable objects and for camera motion: (i) the action network produces a distinct output for each dynamic object in the environment; (ii) (i) + the action and dynamics networks are conditioned on bounding box and camera information; (iii) (ii) + output resolution is increased to match our method; (iv) (ii) + $\mathcal{L}_\Delta$; (v) (iii) + $\mathcal{L}_\Delta$.
Playability evaluation ⟨1⟩. We evaluate player control capabilities in Tab. 3.3. On the *Tennis* dataset our model substantially improves over the baselines in the action space metrics, LPIPS and FVD, suggesting better controllability of the players. In particular, the considerably lower MDR indicates a better capacity of the model in generating players with respect to the baselines. Fig. 3.6 shows qualitative reconstruction results for our method. As suggested by the MDR and ADD scores, the model correctly synthesizes both players and is able to reconstruct the player movements of the ground-truth sequence using only a sequence of discrete actions. In addition, a visualization of the learned action space (see Fig. 3.7) shows that the model learns a set of diverse discrete actions that correspond to the main movement directions. On the *Minecraft* dataset, our method surpasses the baselines both in terms of video and action quality metrics. In particular, the high variation in camera pose in this dataset is not correctly

							Tennis			
	Aux.	*H.Res.*	$\mathcal{L}_\Delta$	LPIPS↓	FID↓	FVD↓	Δ-*MSE*↓	Δ-*Acc*↑	ADD↓	MDR↓
CADDY [85] (i)				0.313	61.0	877	0.901	42.6	35.1	36.9
CADDY [85] (ii)	✓			0.351	69.2	1109	0.592	59.6	29.0	24.8
CADDY [85] (iii)	✓	✓		0.213	15.4	727	0.693	57.5	18.7	11.7
CADDY [85] (iv)	✓		✓	0.445	70.3	1568	0.797	62.4	29.6	33.0
CADDY [85] (v)	✓	✓	✓	0.534	191	8083	0.633	73.5	20.2	60.3
(Ours)				0.181	17.4	485	0.293	95.7	14.0	4.84
							Minecraft			
	Aux.	*H.Res.*	$\mathcal{L}_\Delta$	LPIPS↓	FID↓	FVD↓	Δ-*MSE*↓	Δ-*Acc*↑	ADD↓	MDR↓
CADDY [85] (i)				0.743	288	3667	1.0	(100)	15.1	96.8
CADDY [85] (ii)	✓			0.773	296	2906	0.999	52.8	56.3	93.2
CADDY [85] (iii)	✓	✓		0.677	211	2213	0.998	53.7	42.5	82.9
CADDY [85] (iv)	✓		✓	0.707	275	2553	0.467	83.3	56.2	92.2
CADDY [85] (v)	✓	✓	✓	0.691	309	2187	0.439	82.6	17.4	95.1
(Ours)				0.204	16.8	329	0.271	77.7	17.8	33.9

Table 3.3: Playability evaluation with baselines on the *Tennis* and *Minecraft* datasets and camera control evaluation on the *Minecraft Camera* dataset. *Aux.*: use of auxiliary bounding box and camera pose information; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for Δ-*MSE*. Δ-*MSE*, Δ-*Acc* and MDR in %, ADD in pixels.

modeled by the baseline methods which produce irrealistic results. Note that (i) and (iv) show a better Δ-*ACC* than our method which is explained by their learned action space which only discovers a reduced number of action categories as confirmed by the high Δ-*MSE*.

To further evaluate the quality of the action space we perform a user study (see Tab. 3.5) on the *Tennis* dataset, following the protocol of *Menapace et al.* [85]. We sample a set of 26 frames from the test set and for each frame we produce video continuations conditioned on each learned action. Two separate videos are produced for each initial frame and action, the first cropped on the lower tennis field, the second cropped on the upper tennis field to ensure a single tennis player is depicted in each video. For each produces sequence, we ask 3 *Amazon Mechanical Turk* users to choose which is the performed action between a set of options. We then measure agreement between the users using the Fleiss' kappa measure [26] and compute diversity in the generated actions using entropy of user-selected actions. We consider only the baselines with the highest video quality and with the best action space metrics. Our model achieves the best agreement in terms of Fleiss' kappa measure [26] and comparable diversity to the baselines indicating that the model generates videos that are consistently conditioned by the action and that no mode collapse of the learned actions is present.

Camera control evaluation ⟨2⟩. We evaluate the quality with which the model can synthesize novel views. We initially perform a quantitative evaluation on the *Minecraft Camera* dataset since novel view ground truth is present. We start from the first frame and reconstruct each sequence using the camera parameters of the novel views. Results are shown in Tab. 3.4. Despite the presence of auxiliary bounding box and camera pose inputs for CADDY [85], the baseline method fails in synthesizing the scene from novel perspectives. We ascribe this phenomenon to the lack of an explicit model for the camera. Our method instead can successfully synthesize the scene from novel camera perspectives.

We now evaluate camera manipulation capabilities for our method on the *Tennis* dataset. We consider each test sequence and sample two random camera poses by applying noise to

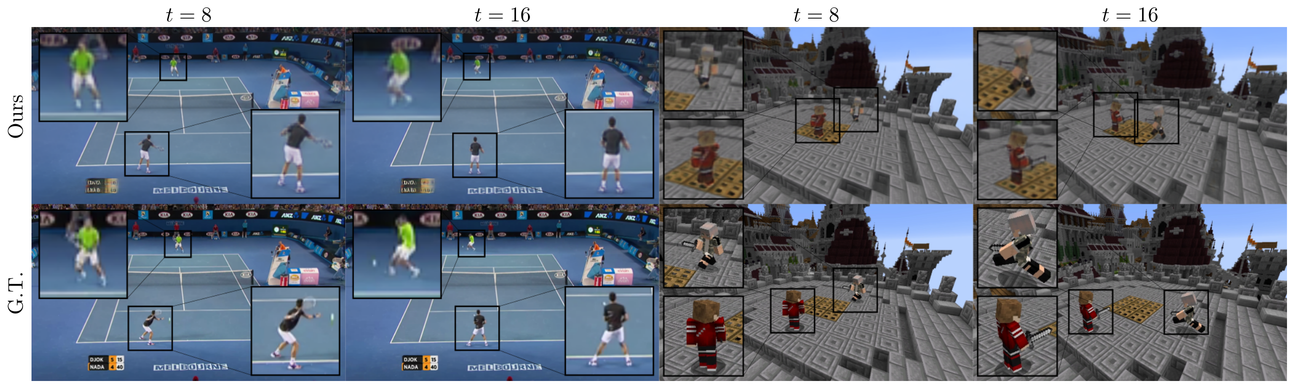


Figure 3.6: Qualitative reconstruction results produced by our method on the *Tennis* and *Minecraft* datasets. In the reconstructed sequence, playable objects move according to the ground truth sequence and are rendered in realistic poses.

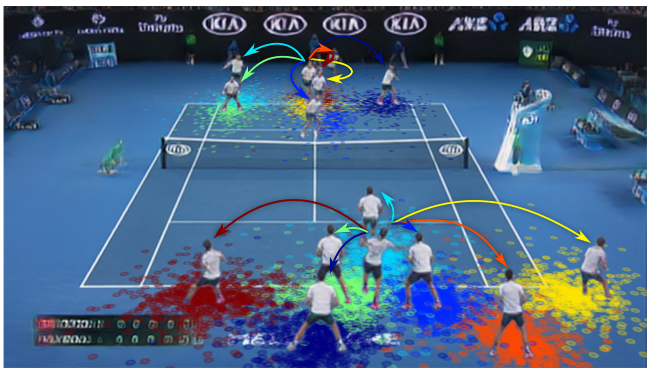


Figure 3.7: Action space learned by our method on the *Tennis* dataset. Each color represents a learned action and each arrow shows the effects of applying the respective action six times to the initial player. The overlay on the floor shows the distribution of possible ending positions after the application of each action.

				Tennis				*Minecraft Camera*			
	Aux.	*H.Res.*	$\mathcal{L}_\Delta$	LPIPS↓	FID↓	ADD↓	MDR↓	LPIPS↓	FID↓	ADD↓	MDR↓
CADDY [85] (i)				0.659	224	50.4	82.0	0.747	306	11.7	95.8
CADDY [85] (ii)	✓			0.623	170	54.7	33.7	0.762	324	44.7	92.2
CADDY [85] (iii)	✓	✓		0.425	26.1	38.1	29.2	0.669	244	29.2	82.0
CADDY [85] (iv)	✓		✓	0.634	164	57.4	41.7	0.699	314	62.0	89.4
CADDY [85] (v)	✓	✓	✓	0.574	216	35.0	66.7	0.679	337	19.1	93.6
(Ours)				0.205	20.5	13.9	5.76	0.242	29.2	5.69	8.07

Table 3.4: Camera control evaluation on the *Tennis* and *Minecraft Camera* datasets. *Aux.*: use of auxiliary bounding box and camera pose information; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for Δ-*MSE*. MDR in %, ADD in pixels.

	Aux.	*H.Res.*	$\mathcal{L}_\Delta$	Agreement↑	Diversity↑	*Other* votes
CADDY [85] (iii)	✓	✓		0.353	1.77	1.86
CADDY [85] (v)	✓	✓	✓	0.170	1.71	24.7
(Ours)				0.444	1.7	0.80

Table 3.5: User study results on the *Tennis* dataset. *Aux.*: use of bounding boxes and camera pose; *H.Res.*: use of the high resolution model; $\mathcal{L}_\Delta$: use of the loss for Δ-*MSE*. *Other* votes in %.

the camera pose parameters in the first frame. We then generate a camera trajectory by interpolating between the two poses and render the sequence from the novel camera trajectory. Note that in the *Tennis* dataset the largest portion of the image is occupied by the field plane. Consequently, we can approximately match each image rendered from the novel trajectory to the corresponding original image by applying a homography. We then compute reconstruction losses between the original and the generated sequence warped according to the corresponding homography. While this evaluation does not account for parts of the image that do not lie on the field plane such as the players, it can be used to detect failure cases. We show the results in Tab. 3.4. LPIPS, ADD and MDR highlight that our method obtains better consistency than the baselines in the generation of novel camera views.

In Fig. 3.8 we show qualitative camera and style manipulation results for our method on the *Tennis* dataset. Our model can synthesize the scene under novel views and correctly alter the style of the field and players to the one of a target image.

3.4.4 Ablation Studies

Synthesis module ablation study ⟨3-6⟩. In this section we evaluate the contribution of each proposed architecture component for the synthesis module: *Multi* use of multi-object modeling ⟨**3**⟩, π use of deformation modeling ⟨**4**⟩, w use of style modulation layers for appearance changes ⟨**5**⟩, F use of the feature renderer for robustness ⟨**6**⟩. We produce the following method variations: (a) no component is used; this approach resembles NeRF [89]; (b) *Multi*; (c) *Multi* and π; this architecture is akin to NR-NeRF [134] with ⟨**3**⟩; (d) *Multi*, π, and w injected with concatenation rather than style modulation layers; (e) *Multi*, π, and w with style modulation layers; (f) *Multi*, π, w and a simplified ConvNet F that renders the complete frame from feature maps at a single resolution; from an architectural viewpoint, this feature rendering strategy resembles the one of GIRAFFE [95].

Results are shown in Tab. 3.6. (c) and (e) show that deformation and style modeling with

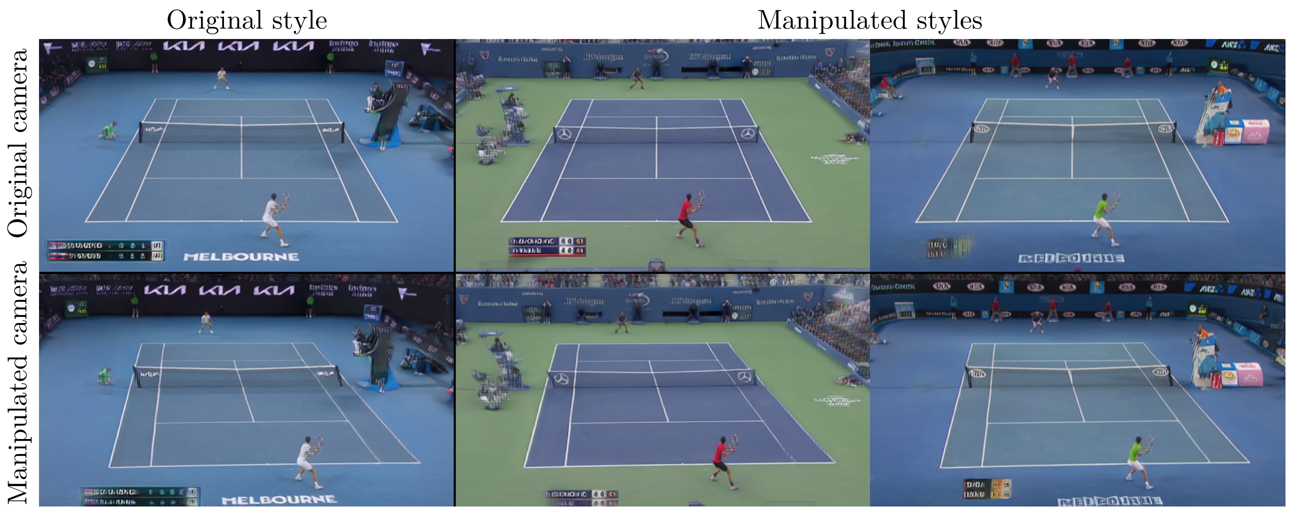


Figure 3.8: Camera and style manipulation results on the *Tennis* dataset. The original image is rendered under a novel camera perspective using varying styles for the field and players.

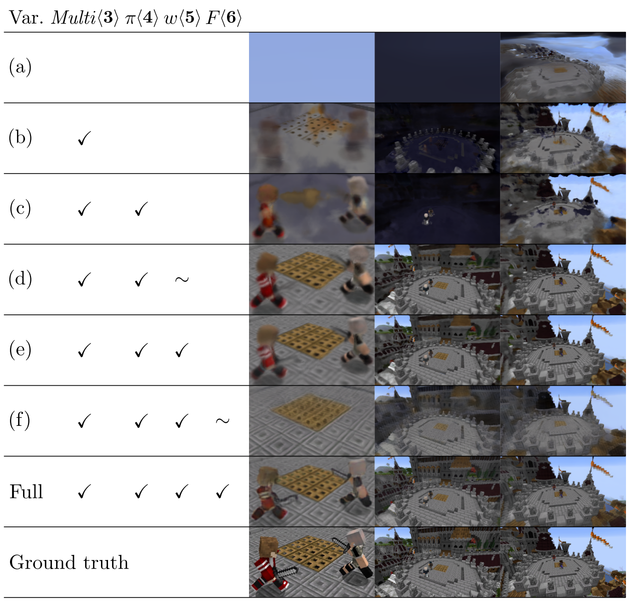
style modulation layers are both necessary to accurately synthesize the scene, but generate blurry results due to calibration and localization errors. We recover sharpness by introducing our ConvNet feature renderer which reduces blur by modeling cross-pixel correlations. Substituting our renderer with the one of (f) leads to performance degradation due to the excessively sparse sampling of rays imposed by memory constraints when rendering the complete frame that leads to 3D consistency artifacts which are particularly apparent in the region of dynamic objects.

We present qualitative results in Fig. 3.9. The evaluation confirms the importance of the use of style modulation layers to model appearance changes and of the bending network to model deformations. In addition, without our feature renderer F, the model produces blurry results and lacks details such as the wrinkles on the clothes. Introducing the feature renderer F allows the model to be trained on complete patches applying the perceptual loss term. This, in conjunction with the ability of ConvNets to model inter-pixel relationships, enables the model to reduce blur and to generate clothing that contains more detailed wrinkles. We also note that, in some sequences, the feature renderer is capable of generating realistic player shadows in portions of the image that lie outside of the bounding volume β for the radiance field of the player. We ascribe this to the capacity of the ConvNet F to model the correlation between the presence of the player and of its shadow.

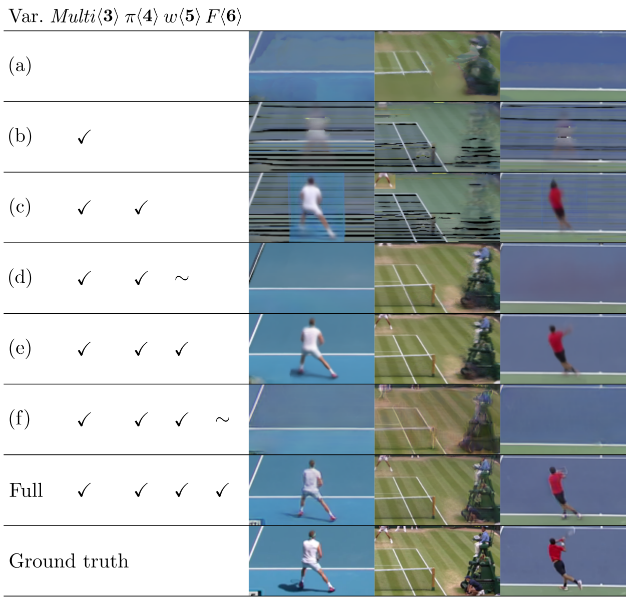
Action module ablation study. We now evaluate the contribution of the main components of the action module by ablating the following: *Rel.* use of camera-relative object movement in the dynamics network; D use of the temporal discriminator; $\mathcal{L}_\Delta$ use of the loss on Δ-MSE; $\mathcal{L}_{\text{act}}$ use of the information-theoretic action learning loss. Results are shown in Tab. 3.7. Removing the temporal discriminator causes an increase in the FVD. A qualitative analysis of the results shows that models not using D produce sequences where the players translate in the scene, but fail to realistically move their limbs. In addition, the introduction of $\mathcal{L}_\Delta$ produces a positive impact on the action space metrics. We also note that, thanks to the presence of $\mathcal{L}_\Delta$, the model learns an action space even in the absence of $\mathcal{L}_{\text{act}}$. Lastly, without camera-relative object movement in the dynamics network, the model produces movements that are independent from the current camera orientation, which is undesirable (Sec. 3.3.2).

Var.	*Multi* ⟨**3**⟩	π ⟨**4**⟩	w ⟨**5**⟩	F ⟨**6**⟩	LPIPS↓	FID↓	FVD↓	ADD↓	MDR↓
						Tennis			
(a)					0.505	256.1	7011	72.2	85.6
(b)	✓				0.619	271.4	10697	3.57	100.0
(c)	✓	✓			0.624	253.5	7280	2.65	28.6
(d)	✓	✓	∼		0.319	96.3	3303	79.8	85.4
(e)	✓	✓	✓		0.286	39.3	638	3.11	7.34
(f)	✓	✓	✓	∼	0.272	39.2	2678	46.9	60.5
Full	✓	✓	✓	✓	0.167	17.1	497	(1.70, 3.41)	(1.61, 6.18)
						Minecraft			
Var.	*Multi* ⟨**3**⟩	π ⟨**4**⟩	w ⟨**5**⟩	F ⟨**6**⟩	LPIPS↓	FID↓	FVD↓	ADD↓	MDR↓
(a)					0.735	376	2548	109.1	99.9
(b)	✓				0.595	266	1617	45.4	86.4
(c)	✓	✓			0.648	301	1818	10.17	50.2
(d)	✓	✓	∼		0.361	68.6	482	7.39	31.9
(e)	✓	✓	✓		0.350	61.0	465	8.27	31.8
(f)	✓	✓	✓	∼	0.341	67.4	1371	88.5	88.8
Full	✓	✓	✓	✓	0.193	16.5	289	5.45	33.7

Table 3.6: Synthesis module ablation results on the *Tennis* and *Minecraft* datasets. *Multi*: use of multi-object modeling, π: use of deformation, w: use of style modulation layers or of direct style encoding (∼), F: use of the feature renderer or of the simplified renderer (∼). ADD in pixels, MDR in %.



(a) *Minecraft*



(b) *Tennis*

Figure 3.9: Synthesis module reconstruction results on the *Minecraft* and *Tennis* dataset. The image is cropped for better visualization. *Multi*: use of multi-object modeling, π: use of deformation, w: use of style modulation layers or of direct style encoding (∼), F: use of the feature renderer or of the simplified renderer (∼).

Var.	*Rel.*	*D*	$\mathcal{L}_\Delta$	$\mathcal{L}_{\text{act}}$	LPIPS↓	FID↓	FVD↓	Δ-*MSE*↓	Δ-*Acc*↑	ADD↓	MDR↓
(A)				✓	0.205	17.0	334	0.903	33.9	18.7	33.0
(B)	✓			✓	0.204	17.0	329	0.290	76.0	18.6	33.5
(C)	✓		✓	✓	0.203	16.9	340	0.263	80.0	15.4	34.0
(D)	✓	✓		✓	0.204	17.0	323	0.289	77.0	17.8	34.3
(E)	✓	✓	✓		0.204	16.9	335	0.276	77.5	17.5	34.0
Full	✓	✓	✓	✓	0.204	16.8	329	0.271	77.7	17.8	33.9

Table 3.7: Action module ablation results on the *Minecraft* dataset. *Rel.*: use of camera relative residual Δ output, *D*: use of the temporal discriminator, $\mathcal{L}_\Delta$: use of the loss for Δ-*MSE*, $\mathcal{L}_{\text{act}}$: use of the information-theoretic action learning loss. Δ-*MSE*, Δ-*Acc* and MDR in %, ADD in pixels.

3.5 Conclusions

In conclusion, in this chapter we present a new framework for the construction of Playable Environments where multiple agents can be controlled through discrete action, the camera can be controlled explicitly, and style of each element in the scene can be controlled. Our framework features an encoder-decoder architecture based on composable NeRFs. An upsampling CNN and a patch-based training procedure are introduced to allow efficient generation of high resolution videos. We introduce an action module to model the dynamics of the environment and create compelling playable environments. An action consistency loss and temporal discriminators are added to improve action control and realism of the generated dynamics. Extensive experimental evaluation on two large-scale datasets shows that this explicit 3D representation, at the cost of camera calibration and object detections, allows the proposed framework to tackle datasets with camera motion and multiple agents that are challenging for 2D methods such as the one described in Chapter 2.

Our method enables the creation of compelling playable experiences and the generation of videos with high degree of control. We argue, however, that interaction with the environment misses important aspect that would be expected from a video game. First, actions must be specified for both players, making it necessary to play against another human player. Second, important elements of gameplay such as the ball in tennis are missing due to the lower resolution at which the synthesis model operated before the upsampler. Lastly, control over the actions of players is coarse and makes it not possible to specify actions such as hitting the ball or the particular tennis shot to perform, important aspects of playability. Addressing such issues requires learning a model of games embued with a deep understanding of the environment and of viable strategies, and capable of autonomously devising actions to reach high-level goals, such as winning a match, a component typically referred to as “game AI”. In the next chapter, we introduce Promptable Game Models to learn such capabilities.

Chapter 4

Promptable Game Models

Game engines are powerful tools in computer graphics. Their power comes at the immense cost of their development. While Chapter 2 and Chapter 3 describe automated methods for the creation of playable experiences with no or limited supervision, we argue that several aspects in the generated playable experiences hinder them from being considered real videogames. The generated experiences, in fact, enable the user to control agents using only a restricted set of discrete actions, whose meaning is assigned by the model itself. This hinders the capability of game developers of deciding how to map actions to game controllers and the user from generating finer, but relevant actions such as hitting a tennis ball with a certain type of shot and deciding where to send the ball. In addition, agent intelligence, or "game AI" is not captured, resulting in the impossibility for users to play against an opponent, an important part of videogames, or for non-playable characters to perform basic tasks such as navigation in the environment.

In this chapter, we present a framework to train controllable models of games that can be prompted with a wide range of conditioning signals and reason on the given constraints to produce realistic outputs showcasing intelligent agent behavior. The result—a Promptable Game Model (PGM)—maintains the state of the scene and its agents and makes it possible for a user to *play* the game by specifying both high- and low-level action sequences. Most captivatingly, our PGM unlocks the *director's mode*, where the game is played by specifying goals for the agents in the form of *prompts* containing *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. We showcase masked transformers trained with a diffusion objective to be proficient at this task. To render the resulting state of the environment and its agents, we use a compositional NeRF representation encapsulated in our synthesis model. Unlike previous works, our compositional NeRF representation makes use of efficient representations of neural radiance fields to enable generation of high-resolution videos without the need for upsampler networks that we found responsible for grid artifacts in the output videos. 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 PGM unlocks applications beyond the capabilities of the current state of the art. To stimulate future research in this direction, we publicly release our code and data.

4.1 Introduction

Recent video generation methods, thanks to their training on extensive web-scale datasets [116], exhibit a remarkable capacity for generating a vast amount of different concepts and scenes [6, 41, 123]. Despite this, their generic nature hinders their comprehension of the dynamics of

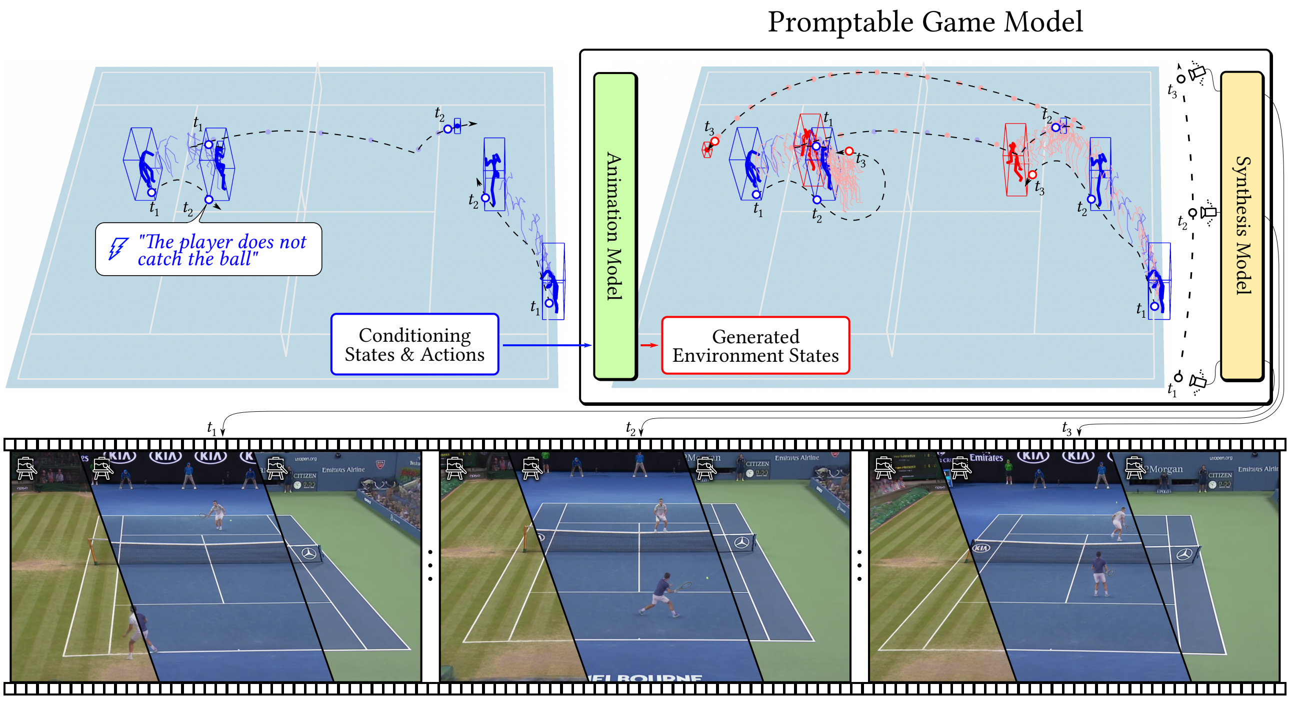

Promptable Game Model
"The player does not catch the ball"
Conditioning States & Actions
Animation Model
Generated Environment States
Synthesis Model
t_1
t_2
t_3

Figure 4.1: We propose Neural Game Engines (NGEs), game-engine-like models that are learned from annotated videos. Our NGE enables the generation of videos using a wide spectrum of conditioning signals such as player poses, object locations, and detailed textual actions (see ⚡) 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 dynamics. 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, our model renders the scene from a user-defined viewpoint (see 🎥) using a *Synthesis Model* where the style of the scene (see 🪑) can be controlled explicitly.

the modeled scenes. When generating or editing videos of a game, such as a tennis match, this limitation impedes their ability to attain precise control of the player movements or to devise optimal strategies to reach desired states of the game, such as victory over the opponent.

Conversely, humans excel at comprehending these dynamics and, after observing a few instances of gameplay, understand the rules that govern the game, the possible actions of the observed players and their effects. This learning process generates an internalized model of the environment that enables the resolution of a range of *prompts*, *i.e.* to devise a game sequence respecting a set of *conditions* dictated by the prompt. Not only this empowers us to envision the evolution of the environment given different courses of action, but also facilitates reasoning on which actions should be taken to best respond to the opponent or which path an agent should take to reach a designated point. Learning a generative model capable of answering such queries would unlock video generation and editing methods capable of achieving a higher level of semantic control over the generation process thanks to their understanding of the scene dynamics. In this context, the question emerges: *Can we learn a model with such abilities?*

Neural video game simulators, a growing category of video generation methods, make an important step in this direction by focusing on modeling the dynamics of an environment, often a sports or computer game, with high fidelity and degree of control, and show that annotated videos can be used to learn to generate videos interactively [21, 47, 55, 57, 85] and build 3D environments where agents can be controlled through a set of discrete actions [86]. However, when applied to complex or real-world environments, these works present several limitations: they do not accurately model the game's dynamics, do not model physical interactions of objects in 3D space, do not learn precise controls, do not allow for high-level goal-driven control of the game flow, and, finally, do not model intelligent behavior of the agents, a capability often referred to as "game AI".

In this work, we overcome these limitations by introducing game models trained on a set of annotated videos that support complex prompts. Due to the versatility of the applications enabled by diverse prompting methods (see Sec. 4.4), we call them Promptable Game Models (PGMs). More formally, we define PGMs as those models supporting a core set of game modeling and prompting functions including rendering from a controllable viewpoint, modeling of game's dynamics, precise character control, high-level goal-driven control of the game, and game AI. Making a first step towards the realization of such models, we propose a framework that supports these characteristics.

To overcome the limitations of [21, 47, 55, 57, 85, 86], 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 dynamics of the game and game AI—important parts of PGMs—and that such commentaries are a key component in enabling a series of important model capabilities related to precise character control and high-level goal-driven control of the game flow.

In its simplest form, for games like Minecraft, this allows the user to instruct the player to perform sequences of actions such as *"Jump onto a birch pole and run through the stairs"*, while for tennis, this enables controlling each player in a precise manner with instructions such as *"hit the ball with a backhand and send it to the right service box"*. Training with language enables our model to deeply understand semantic parts of the environment in which the game is played. For example, the model learns the locations of certain *parts* of the environment, as well as the sequence of actions necessary to end up in these locations. In tennis, our PGM understands the locations of the left & right service boxes, no-man's land, and so on. Interestingly, these inferences are made from language and language alone—a remarkable finding also highlighted in PartGlot [62].

Moreover, language enables users to take the *director's mode* and prompt the model with high-level game-specific scenarios or scripts, specified by means of *natural language* and *de-*

sired states of the environment. As an example, given desired starting and ending states, our promptable game model can devise in-between scenarios that led to the observed outcome. Most interestingly, as shown in Fig. 4.1, given the initial states of a real tennis video in which a player lost a point, our model 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 [19, 127, 128], 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 dynamics. Our framework follows this high-level structure highlighted in Fig. 4.1. 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 of [86] followed by a learnable enhancer for superior rendering quality. To model the dynamics of games and game AI that determine the evolution of the environment states, we introduce an animation model. Specifically, inspired by [36], we train a *non-autoregressive text-conditioned* diffusion model which leverages masked sequence modeling to express the conditioning signals corresponding to a prompt. In particular, we show that using text labels describing actions happening in a game is instrumental in learning such capabilities. While certain prior work [55, 57, 85, 86] explored maintaining and rendering states of games, we are not aware of any generative method that attempts to enable precise control, modeling sophisticated goal-driven game dynamics, 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, 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 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 framework for the creation of Promptable Game Models.** It supports detailed offline rendering of high-resolution, high-frame rate videos of scenes with articulated objects from controllable viewpoints. It can generate actions specified by detailed 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 backed by an efficient plane- and voxel-based object representation that operates without upsampling. With respect to the upsampler-based approach of [86], it doubles the output resolution, can synthesize small objects and does not present checkerboard upsampling artifacts.

- **An animation model**, based on a text-conditioned diffusion model with a masked training procedure, which is key to supporting complex game dynamics, object interactions, game AI, and understanding detailed actions. It unlocks applications currently out of reach of state-of-the-art neural video game simulators (see Sec. 4.4).

- **Two datasets**: a large-scale 15h Tennis and a 1h Minecraft video datasets with camera calibration, 3D player poses, 3D ball localization, and detailed text captions.

4.2 Related Work

Our Promptable Game Model relates to neural game simulation literature, game engines, character animation, neural rendering, sequential data generation, and text-based generation. We review the most recent related works in this section.

4.2.1 Neural video game simulation

In the last few years, video game simulation using deep neural networks has emerged as a new research trend [21, 47, 55, 57, 85, 86]. The objective is to train a neural network to synthesize videos based on a specific type of prompt: a sequence 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 [16,57,98]. They consider a discrete action representation that is difficult to define a priori for real-world environments. More recently, [55] 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, [85,86] propose to learn a discrete action representation. [47] expands on this idea by modeling actions as a learned set of geometric transformations, while [21] represents actions by separating them into a global shift component and a local discrete action component.

Differently from our PGM, previous works perform generation in an autoregressive manner, conditioned on the actions and, therefore, are unable to answer prompts entailing constraint- or goal-driven generation for which non-sequential conditioning is necessary. We find the proposed text-based action representation and masked training procedure to be crucial to unlocking such applications.

Among these works, *Playable environments* [86] is the most closely related to ours. Rather than employing a 2D model, they use a NeRF-based renderer [89] that enables them to represent complex 3D scenes. We follow this high-level design but introduce a more efficient plane- and voxel-based NeRF representation that enables the rendering of outputs at double the original resolution without the use of upsampling modules which we found to be the cause of checkerboard artifacts, failures in rendering of small objects and to be prone to failure when training at higher resolutions. In addition, 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. Lastly, we replace the adversarially-trained LSTM animation module with a more capable masked diffusion transformer.

4.2.2 Game Engines

Game engines brought a revolution to game development by providing extensible and reusable software that can be employed to create a wide range of game models [35]. Nowadays, a range of game engines exists (*Unity*, *Unreal*, *id Tech*, *Source*, *CRYENGINE*, *Frostbite*, *RAGE*) and have grown to become vast software ecosystems. Modern game engines are organized into components including a rendering engine [93], a resource manager, a module for physics and collision, an animation manager and, importantly, a gameplay foundation system that models the game rules and encapsulates game AI functionalities [35]. The presence of these components, coupled with the labor of a range of trained experts including software engineers, artists (animators, 3D modelers, texture and lighting artists) and game developers, enables the construction of sophisticated game models supporting low-level character control and scripted agent behavior. We show that monocular videos annotated with a fraction of the effort can be used to learn models of games that support answering challenging prompts related to agent intelligence, a capability difficult to achieve through scripted agent behavior.

4.2.3 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.* [44] propose a learnable version of Motion Matching [8] 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 [45, 71, 128, 129]. Starke *et al.* [128] propose a model based on a mixture of experts that controls character locomotion and object interactions, in a follow-up work [129] 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 [44,128,129], motion phases [128] or engineered action labels [128, 129]. 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 requirements by not using motion capture and basing user control on natural language that is cheaper to acquire 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 [128]. In contrast, our method models complex game AI tasks such as modeling strategies to defeat the opponent, which are instrumental in answering complex user prompts.

4.2.4 Neural Rendering

Neural rendering was recently revolutionized by the advent of NeRF [89]. Several modifications of the NeRF framework were proposed to model deformable objects [76,100,101,134,143], and decomposed scene representations [67, 86, 91, 95, 99, 144]. In addition, several works improved the efficiency of the original MLP representation of the radiance field [89] by employing octrees [82, 147], voxel grids [31], triplanes [11], hash tables [92], or factorized representations [13].

Our framework is most related to that of [143], since we model player deformations using an articulated 3D prior and linear blend skinning (LBS) [73]. Differently from 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 [86], our framework does not adopt computationally-inefficient MLP representations, using voxel [31] or plane representations instead, thus does not rely on upsampler networks.

4.2.5 Sequential data generation with diffusion models

In prior work, sequential data generation was mainly addressed with auto-regressive formulations combined with adversarial [68] or variational [4, 28] generative models. Recently, diffusion models have emerged as a promising solution to this problem leading to impressive results in multiple applications such as audio [14, 61, 69, 72] and video synthesis [6, 41, 43, 123], language modeling [22], and human motion synthesis [20, 153]. Following this methodological direction [131], introduces a score-based diffusion model for imputing missing values in time series. They introduce a training procedure based on masks that simulate missing data. This approach motivates our choice of a similar masking strategy to model the conditions entailed by the given prompt and 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.

4.2.6 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 [107, 108, 111, 114] and videos with arbitrary content [41, 43, 46, 123], and arbitrary 3D shapes [2, 49, 77].

Han *et al.* [36] 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 [78] language model and a sequence masking technique. Fu *et al.* [32] 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 [3, 132]. Recently, diffusion models have shown strong performance on this task [20, 153]. 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.

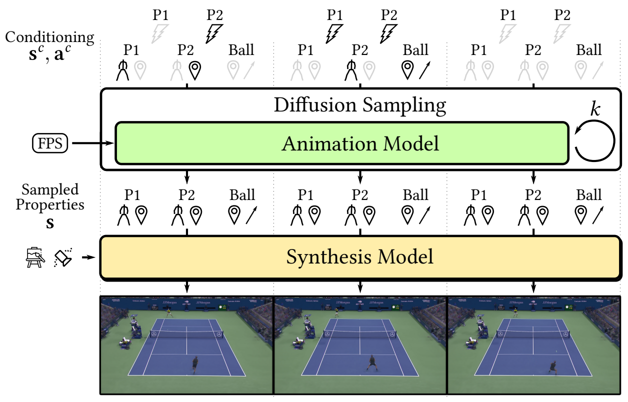
4.3 Method

This section introduces our framework for the creation of *Promptable Game Models* that allows the user to perform a range of dynamic scene editing tasks, formulated as a set of conditioning *prompts*.

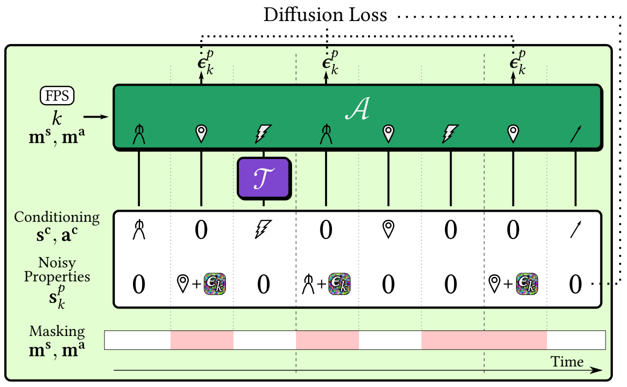
Similarly to game engines that maintain states of each object, render the environment using a graphics pipeline, and have a model of the game dynamics, we divide our PGM into two modules: a *synthesis model* and an *animation model*. The synthesis model generates an image given the representation of the environment state. The animation model, instead, aims at modeling the game's dynamics, 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. 4.2a.

In more detail, our model 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 $\mathbf{s}_t \in \mathbb{S} = (\mathbb{R}^{n_1} \times ... \times \mathbb{R}^{n_P})$, 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.

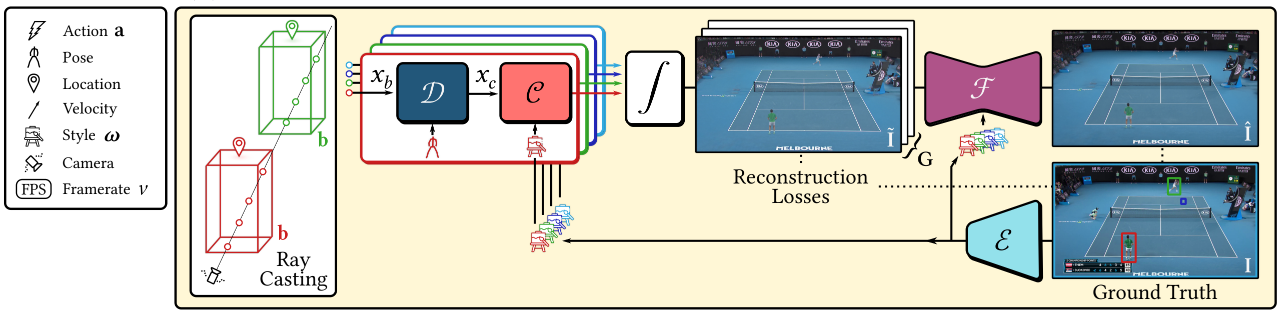
On the other hand, the animation model predicts the evolution of an environment in time, which is represented by the sequence of its states $\{\mathbf{s}_1, \mathbf{s}_2, \dots \mathbf{s}_T\} = \mathbf{s} \in \mathbb{S}^T$, where T is the length of the sequence. The model provides control over sequence generation with the help of user-defined conditioning signals, or prompts, 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



(a) Model Overview



(b) Animation model



(c) Synthesis model

Figure 4.2: (a) Overview of our framework. The *animation model* produces states $\mathbf{s}$ based on user-provided conditioning signals, or *prompts*, $\mathbf{s}^c, \mathbf{a}^c$ that are rendered by the *synthesis model*. (b) The diffusion-based animation model predicts noise $\boldsymbol{\epsilon}_k$ applied to the noisy states $\mathbf{s}_k^p$ conditioned on known states $\mathbf{s}^c$ and actions $\mathbf{a}^c$ with the respective masks $\mathbf{m}^\mathbf{s}, \mathbf{m}^\mathbf{a}$, diffusion step k and framerate ν. 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 $\boldsymbol{\omega}$ of each object. Each object is represented in its canonical pose by $\mathcal{C}$ and deformations of articulated objects are modeled by the *deformation model* $\mathcal{D}$. After integration and composition, the feature grid $\mathbf{G}$ is rendered to the final image using the *feature enhancer* $\mathcal{F}$.

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. 4.7). 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 $\mathbf{s}$ and actions $\mathbf{a}$.

4.3.1 Synthesis Model

In this section, we describe the synthesis model that renders states from controllable viewpoints (see Fig. 4.2c). We build our model based on a compositional NeRF [86] 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 [86]. Our model is trained using reconstruction as the main guiding signal.

In Sec. 4.3.1-4.3.1 we illustrate the main components of the synthesis module and in Sec. 4.3.1 we describe the training procedure.

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 $\mathbf{x}$ and viewing direction $\mathbf{d}$ to density σ and radiance $\mathbf{c}$.

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 [86, 91, 146]. The scene is rendered by sampling points independently for each object and querying the respective object radiance field $\mathcal{C}_i$. The results for all objects are then merged and sorted by distance from the camera before being integrated.

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 $\mathbf{b}_i^{3\text{D}}$ which is defined by size and position. In the case of the ball, we additionally consider its velocity to model blur effects (Sec. 4.3.1).

- *Object style.* All objects have an appearance that may vary in different sequences, thus we introduce a style code $\boldsymbol{\omega}_i$ 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 $\mathbf{R}$ and translation $\mathbf{tr}$ parameters associated with each joint (Sec. 4.3.1).

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

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 $\boldsymbol{\omega}$ for each object as a latent variable that we regress using a convolutional style encoder $\mathcal{E}$. Given the current video frame $\mathbf{I}$ with O objects, we compute 2D bounding boxes $\mathbf{b}^{2\mathrm{D}}$ for each object. First, a set of residual blocks is used to extract frame features which are later cropped around each object according to $\mathbf{b}^{2\mathrm{D}}$ using RoI pooling [34]. Later, a series of convolutional layers with a final projection is used to predict the style code $\boldsymbol{\omega}$ from the cropped feature maps.

Volume Modeling for Efficient Sampling

Radiance fields are commonly parametrized using MLPs [89] 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 $\mathcal{C}$ of each object in a canonical space using two alternative parametrizations.

For three-dimensional objects, we make use of a voxel grid parametrization [31,143]. Starting from a fixed noise tensor $\mathbf{V}' \in \mathbb{R}^{F' \times H'_V \times W'_V \times D'_V}$, a series of 3D convolutions produces a voxel $\mathbf{V} \in \mathbb{R}^{F+1 \times H_V \times W_V \times D_V}$ 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 $\mathbf{x}_c$, the associated features and density σ are retrieved using trilinear sampling on $\mathbf{V}$. To model the different appearance of each object, we adopt a small MLP conditioned on the style $\boldsymbol{\omega}$ to produce a stylized feature with the help of weight demodulation [54].

For two-dimensional objects such as planar scene elements, we make use of a similar parametrization where a fixed 2D noise tensor $\mathbf{P}' \in \mathbb{R}^{F' \times H'_P \times W'_P}$ is mapped to a plane of features $\mathbf{P} \in \mathbb{R}^{F \times H_P \times W_P}$ using a series of 2D convolutions. Given a ray r, we compute the intersection point $\mathbf{x}$ between the plane and the ray which is used to sample $\mathbf{P}$ using bilinear sampling. Similarly to the voxel case, a small MLP is used to model object appearance according to $\boldsymbol{\omega}$. 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.

Deformation Modeling

Since the radiance field $\mathcal{C}$ 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 $[\mathbf{R}_j|\mathbf{tr}_j]$ from each joint $j \in 1, ..., J$ to the parent joint is part of the object's properties. From these we can follow the kinematic tree to derive transformations $[\mathbf{R}'_j|\mathbf{tr}'_j]$ for each joint from the bounding box coordinate system to the canonical coordinate system. Intuitively, these transformations represent how to map a point $\mathbf{x}_b$ in the bounding box coordinate system belonging to the joint j to the corresponding point $\mathbf{x}_c$ in the canonical space.

We implement a deformation procedure based on linear blend skinning (LBS) [73] that establishes correspondences between points in the canonical space $\mathbf{x}_c$ and in the deformed bounding box space $\mathbf{x}_b$ by introducing blending weights $\mathbf{w}$ 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.

$$\mathbf{x}_b = \sum_{j=1}^{J} w_j(\mathbf{x}_c) \left(\mathbf{R}'^{-1}_j \mathbf{x}_c - \mathbf{R}'^{-1}_j \mathbf{tr}'_j \right). \tag{4.1}$$

During volumetric rendering, however, we sample points $\mathbf{x}_b$ in the bounding box space and query the canonical volume in the corresponding canonical space point $\mathbf{x}_c$. Doing so requires solving Eq. (4.1) for $\mathbf{x}_c$, which is prohibitively expensive [76]. Inspired by HumanNeRF [143], instead of modeling LBS weights $\mathbf{w}$, we introduce inverse linear blending weights $\mathbf{w}^b$:

$$\mathbf{w}_j^b(\mathbf{x}_b) = \frac{\mathbf{w}_j(\mathbf{R}_j'\mathbf{x}_b + \mathbf{tr}_j')}{\sum_{j=1}^{J} \mathbf{w}_j(\mathbf{R}_p'\mathbf{x}_b + \mathbf{tr}_j')}. \tag{4.2}$$

such that the canonical point can be approximated as:

$$\mathbf{x}_c = \sum_{j=1}^{J} \mathbf{w}_j^b(\mathbf{x}_b) \left(\mathbf{R}_j'\mathbf{x}_b + \mathbf{tr}_j'\right). \tag{4.3}$$

We parametrize the function $\mathbf{w}$ mapping spatial locations in the canonical space to blending weights as a neural network. Similarly to $\mathcal{C}$, we employ 3D convolutions to map a fixed noise volume $\mathbf{W}' \in \mathbb{R}^{F' \times H_W' \times W_W' \times D_W'}$ to a volume of blending weights $\mathbf{W} \in \mathbb{R}^{J+1 \times H_W \times W_W \times D_W}$, 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 [143].

Enhancer

NeRF models are often parametrized to output radiance $\mathbf{c} \in \mathbb{R}^3$ 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 $\mathcal{C}$ to output features where the first three channels represent radiance and the subsequent represent learnable features. Then, we produce a feature grid $\mathbf{G} \in \mathbb{R}^{F \times H \times W}$ and an RGB image $\tilde{\mathbf{I}} \in \mathbb{R}^{3 \times H \times W}$. We introduce an enhancer network $\mathcal{F}$ modeled as a UNet [112] architecture interleaved with weight demodulation layers [54] that maps $\mathbf{G}$ and the style codes $\boldsymbol{\omega}$ to the final RGB output $\hat{\mathbf{I}} \in \mathbb{R}^{3 \times H \times W}$.

Object-specific rendering

Our compositional approach allows the use of object-specific techniques. In particular, in the case of tennis, 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.

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 [18], 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 $\mathbf{v}$. 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.

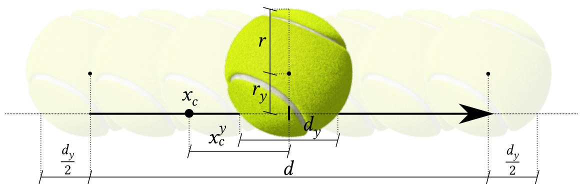


Figure 4.3: Visualization of the quantities involved in the computation of the probability p that the ball intersects with a certain point in space during a randomly sampled time instant in the interval from the opening to the closing of the shutter of the camera to capture the current frame. The leftmost 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.

To compute p (see Fig. 4.3), we first use the velocity vector $\mathbf{v}$ to estimate the rotation $\mathbf{R}_b$ that maps each point $\mathbf{x}_b$ in the ball bounding box to a canonical space $\mathbf{x}_c$ in which the ball velocity vector is aligned to the positive y-axis $\mathbf{x}_c = \mathbf{R}_b\mathbf{x}_b$. Then we compute the distance traveled by the ball while the shutter remains open $d = ||\mathbf{v}||_2 t_c$. We then compute the useful cross-section of the ball d_y that can intersect with $\mathbf{x}_c$ 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 $\mathbf{x}_c$ from the y-axis:

$$d_y = \begin{cases} 2r\sin\left(\arccos\left(\frac{r_y}{r}\right)\right) & \text{if } r_y \leq r \\ 0 & \text{otherwise} \end{cases}. \tag{4.4}$$

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 $\mathbf{x}_c$:

$$p(\mathbf{x}_c) = \max\left(0, \min\left(\min\left(\frac{d_y}{d}, 1\right), \frac{1}{2} + \frac{d_y}{2d} - \frac{|\mathbf{x}_c^y|}{d}\right)\right), \tag{4.5}$$

where $\mathbf{x}_c^y$ is the y-axis coordinate of $\mathbf{x}_c$.

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 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 $\mathbf{W}$ for the desired joint to have a high value in the region corresponding to the mesh (see Eq. (4.2)).

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. 4.4 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

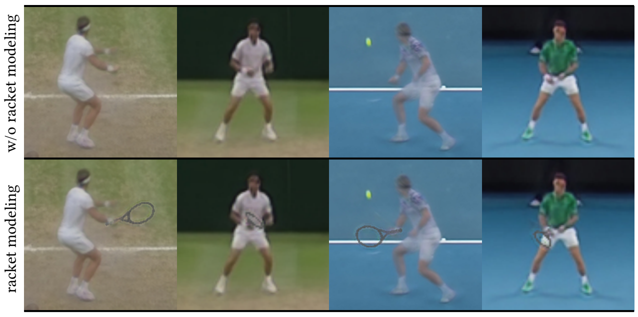


Figure 4.4: Examples of tennis scenes with and without inserted rackets.

the enhancer rather than the enhanced image $\hat{\mathbf{I}}$ for pixels corresponding to the inserted object and its contour region.

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 [86]. 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.

Skybox Modeling The *Minecraft* background is represented as a skybox that is modeled by extending the planar object modeling mechanism of Sec. 4.3.1. In more detail, we sample the feature plane $\mathbf{P}$ 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.

Training

We train our model using reconstruction as the main driving signal. Given a frame $\mathbf{I}$ and reconstructed frame $\hat{\mathbf{I}}$, we use a combination of L2 reconstruction loss and the perceptual loss of Johnson *et al.* [53] as our training loss. To minimize the alterations introduced by the enhancer and improve view consistency, we impose the same losses between $\mathbf{I}$ and $\tilde{\mathbf{I}}$, 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 [86].

We train all the components of the synthesis model jointly using Adam [59] 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.

4.3.2 Animation Model

In this section, we describe the animation model (see Fig. 4.2b), whose task is that of generating sequences of states $\mathbf{s} \in \mathbb{S}^T$ according to user inputs. The animation model allows users to specify conditioning signals, or prompts, 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 $\mathbf{s}^c \in \mathbb{S}^T$. 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 $\mathbf{a}^c \in \mathbb{L}^{A\times T}$ 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 $\mathbf{s}^c$ and $\mathbf{a}^c$ to be optional, thus we introduce their respective masks $\mathbf{m}^{\mathbf{s}} \in \{0,1\}^{P\times T}$ and $\mathbf{m}^{\mathbf{a}} \in \{0,1\}^{A\times T}$

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 $\mathbf{s}^p \in \mathbb{S}^T$ conditioned on $\mathbf{s}^c$ and $\mathbf{a}^c$ such that:

$$\mathbf{s} = \mathbf{s}^p + \mathbf{s}^c, \tag{4.6}$$

where we consider the entries in $\mathbf{s}^p$ and $\mathbf{s}^c$ to be mutually exclusive, *i.e.* an element of $\mathbf{s}^p$ is 0 if the corresponding conditioning signal in $\mathbf{s}^c$ is present according to $\mathbf{m}^\mathbf{s}$. Note that the prediction of actions is not necessary, since $\mathbf{s}$ 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 pretrained language model in a text encoder $\mathcal{T}$ to model action conditioning information [36]. The masked design provides support for the optional conditioning signals and is trained using masked sequence modeling, where we sample $\mathbf{m}^\mathbf{s}$ and $\mathbf{m}^\mathbf{a}$ according to various strategies that emulate desired inference tasks.

In Sec. 4.3.2 we define our text encoder, Sec. 4.3.2 defines the diffusion backbone, and in Sec. 4.3.2 we describe the training procedure.

Text Encoder

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

$$\mathbf{a}^{\text{emb}} = \mathcal{T}(\mathbf{a}^c) \in \mathbb{R}^{A \times T \times N_t}, \tag{4.7}$$

where N_t is the size of the embedding for the individual sentence. Given a textual action, we leverage a pretrained T5 text model [106] $\mathcal{T}_{\text{enc}}$ that tokenizes the sequence and produces an output feature for each token. Successively, a feature aggregator $\mathcal{T}_{\text{agg}}$ modeled as a transformer encoder [137] produces the aggregated text embedding from the text model features. To retain existing knowledge into $\mathcal{T}_{\text{enc}}$, we keep it frozen and only train the feature aggregator $\mathcal{T}_{\text{agg}}$.

Temporal Modeling

In this section, we introduce the temporal model $\mathcal{A}$ that predicts the sequence of states $\mathbf{s}$ conditioned on known state values $\mathbf{s}^c$, action embeddings $\mathbf{a}^{\text{emb}}$, and the respective masks $\mathbf{m}^\mathbf{s}$ and $\mathbf{m}^\mathbf{a}$. Since only unknown state values need to be predicted, the model predicts $\mathbf{s}^p$ and the complete sequence of states is obtained as $\mathbf{s} = \mathbf{s}^p + \mathbf{s}^c$, following Eq. (4.6). Diffusion models have recently shown state-of-the-art performance on several tasks closely related to our setting such as sequence modeling [131] and text-conditioned human motion generation [20,153]. Thus, we follow the DDPM [42] diffusion framework, and we frame the prediction of $\mathbf{s}^p = \mathbf{s}^p_0$ as a progressive denoising process $\mathbf{s}^p_0, ..., \mathbf{s}^p_K$, where we introduce the diffusion timestep index $k \in 0, ..., K$. The temporal model $\mathcal{A}$ acts as a noise estimator that predicts the Gaussian noise $\boldsymbol{\epsilon}_k$ in the noisy sequence of unknown states $\mathbf{s}^p_k$ at diffusion timestep k:

$$\boldsymbol{\epsilon}^p_k = \mathcal{A}(\mathbf{s}^p_k | \mathbf{s}^c, \mathbf{a}^{\text{emb}}, \mathbf{m}^\mathbf{s}, \mathbf{m}^\mathbf{a}, k). \tag{4.8}$$

An illustration of the proposed diffusion model is shown in Fig. 4.2b.

We realize $\mathcal{A}$ using a transformer encoder [137]. 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 $\mathbf{s}^p_k$ and $\mathbf{s}^c$ 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 $\mathbf{a}^{\text{emb}}$ and, subsequently, the projection outputs for states and actions are concatenated into a single sequence $\mathbf{e} \in \mathbb{R}^{P + A \times T \times E}$, which constitutes the input to the transformer. An output projection layer with separate weights for each object property produces the prediction $\boldsymbol{\epsilon}^p_k$ at the original dimensionality. To condition the model on the diffusion time-step k, we introduce a weight demodulation layer [54] after each self-attention and feedforward block [153].

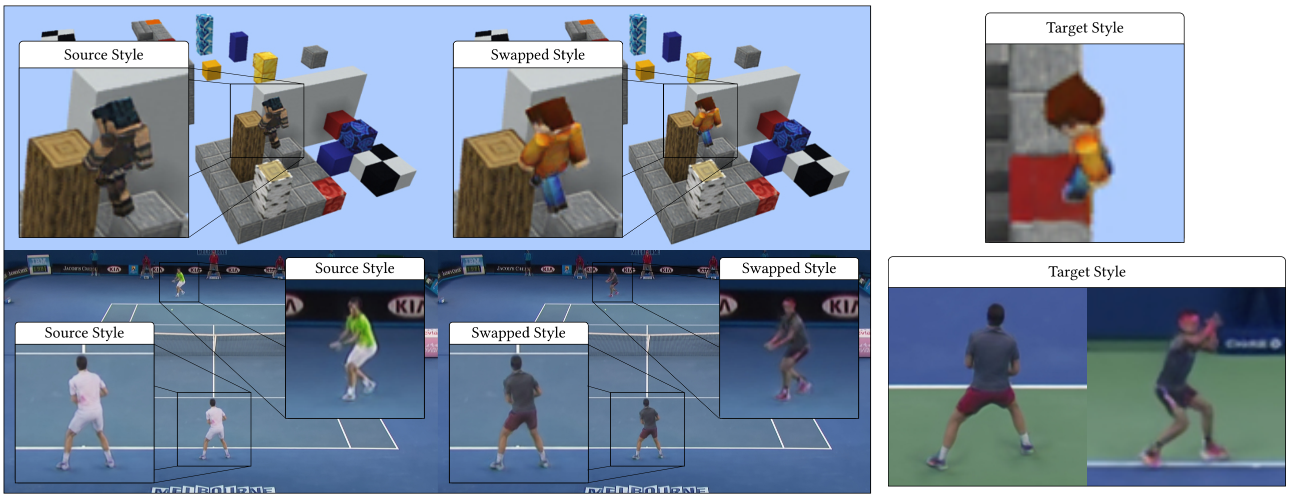


Figure 4.5: Style swap results on the *Tennis* and *Minecraft* datasets. We produce the central image by swapping the style code $\boldsymbol{\omega}$ for the players on the leftmost image with the ones from the rightmost image. *Minecraft* results are cropped for better visualization.

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 ν to enable a progressive increase of the framerate at inference time (see Sec. 4.5.4).

Training

To train our model, we sample a sequence $\mathbf{s}$ with corresponding actions $\mathbf{a}$ from a video in the dataset at a uniformly sampled framerate ν. Successively, we obtain masks $\mathbf{m}^{\mathbf{s}}$ and $\mathbf{m}^{\mathbf{a}}$ according to masking strategies we detail in Sec. 4.5.3. The sequence for training are obtained following $\mathbf{s}_0^p = \mathbf{s} \odot (1 - \mathbf{m}^{\mathbf{s}})$ and $\mathbf{s}^c = \mathbf{s} \odot \mathbf{m}^{\mathbf{s}}$, and actions as $\mathbf{a}^c = \mathbf{a} \odot \mathbf{m}^{\mathbf{a}}$, where $\odot$ denotes the Hadamard product.

We train our model by minimizing the DDPM [42] training objective:

$$\mathbb{E}_{k \sim \mathcal{U}(1,K), \epsilon \sim \mathcal{N}(0,I)} ||\boldsymbol{\epsilon}_k^p - \boldsymbol{\epsilon}_k||, \tag{4.9}$$

where $\boldsymbol{\epsilon}_k^p$ is the noise estimated by the temporal model $\mathcal{A}$ according to Eq. (4.8). Note that the loss is not applied to positions in the sequence corresponding to conditioning signals [131].

Our model is trained using the Adam [59] 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 [42].

4.4 Applications

Our framework 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 dynamics and can be conditioned on a wide range of signals. In the following, we demonstrate a set of selected applications.

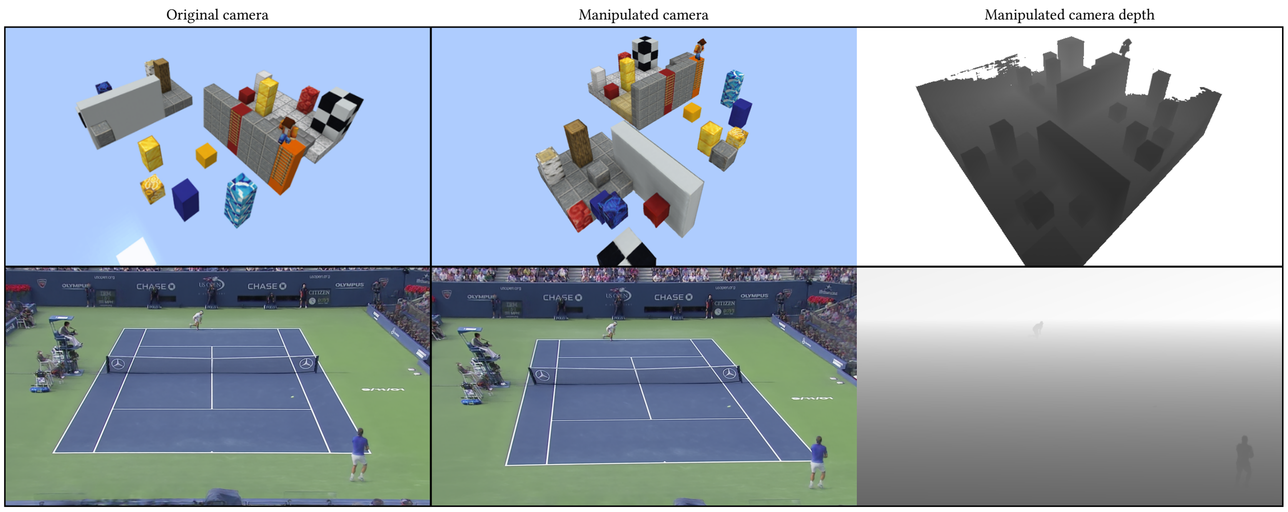


Figure 4.6: Camera manipulation results on the *Tennis* and *Minecraft* datasets. The lack of camera translation on the *Tennis* dataset does not allow to capture the 3D geometry of static objects, which is replaced by a prior [86].

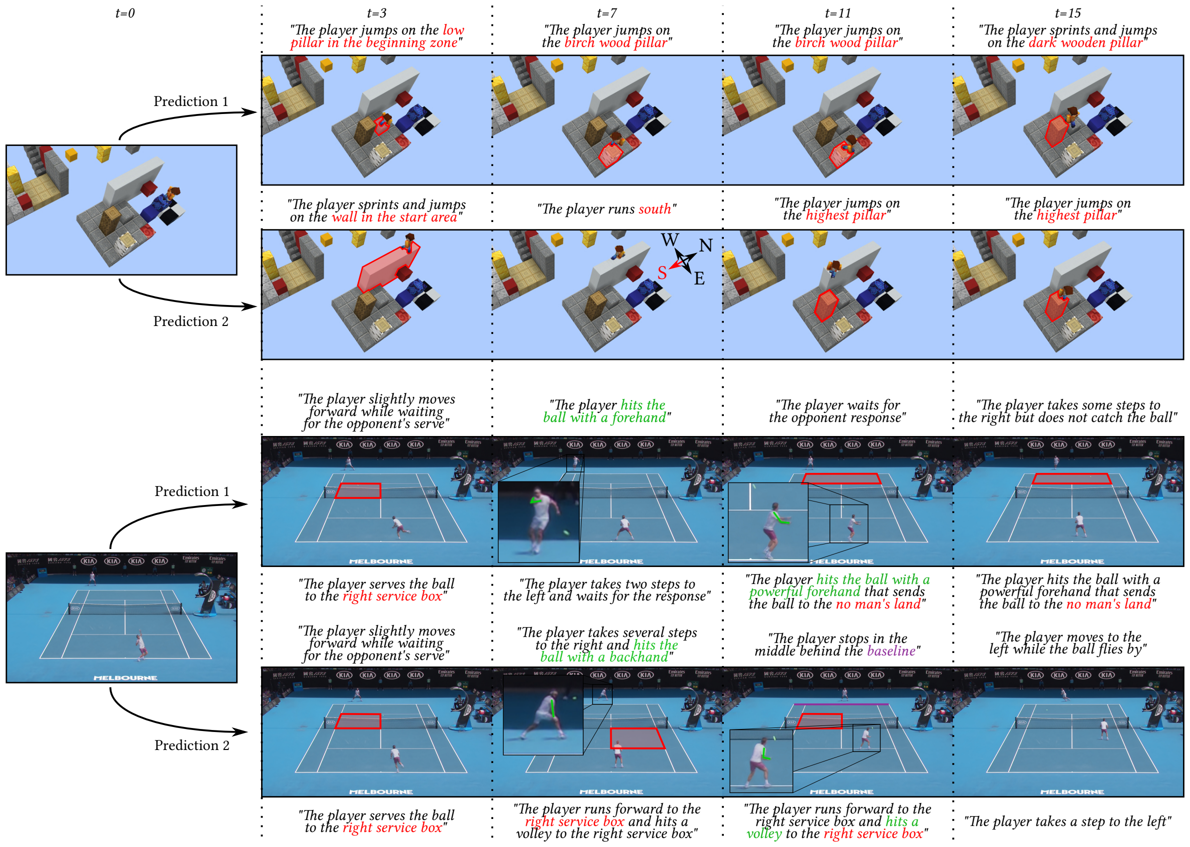

t=0
t=3
"The player jumps on the low pillar in the beginning zone"
t=7
"The player jumps on the birch wood pillar"
t=11
"The player jumps on the birch wood pillar"
t=15
"The player sprints and jumps on the dark wooden pillar"
Prediction 1
"The player sprints and jumps on the wall in the start area"
"The player runs south"
"The player jumps on the highest pillar"
"The player jumps on the highest pillar"
Prediction 2
W
N
S
E
"The player slightly moves forward while waiting for the opponent's serve"
"The player hits the ball with a forehand"
"The player waits for the opponent response"
"The player takes some steps to the right but does not catch the ball"
Prediction 1
"The player serves the ball to the right service box"
"The player takes two steps to the left and waits for the response"
"The player hits the ball with a powerful forehand that sends the ball to the no man's land"
"The player hits the ball with a powerful forehand that sends the ball to the no man's land"
"The player slightly moves forward while waiting for the opponent's serve"
"The player takes several steps to the right and hits the ball with a backhand"
"The player stops in the middle behind the baseline"
"The player moves to the left while the ball flies by"
Prediction 2
"The player serves the ball to the right service box"
"The player runs forward to the right service box and hits a volley to the right service box"
"The player runs forward to the right service box and hits a volley to the right service box"
"The player takes a step to the left"

Figure 4.7: 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”*).

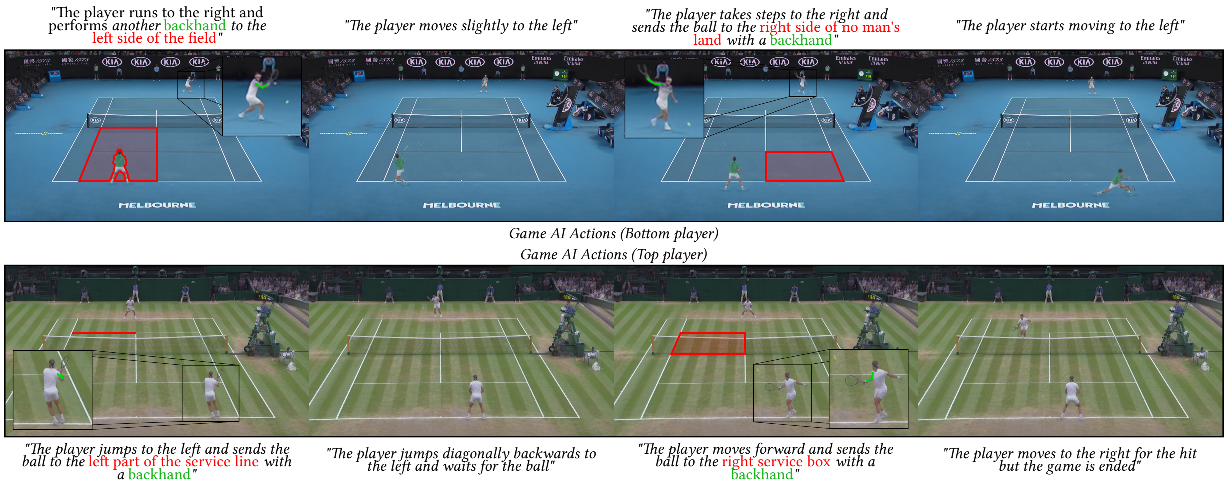

"The player runs to the right and performs another backhand to the left side of the field"
"The player moves slightly to the left"
"The player takes steps to the right and sends the ball to the right side of no man's land with a backhand"
"The player starts moving to the left"
MELBOURNE
Game AI Actions (Bottom player)
Game AI Actions (Top player)
"The player jumps to the left and sends the ball to the left part of the service line with a backhand"
"The player jumps diagonally backwards to the left and waits for the ball"
"The player moves forward and sends the ball to the right service box with a backhand"
"The player moves to the right for the hit but the game is ended"

Figure 4.8: 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.

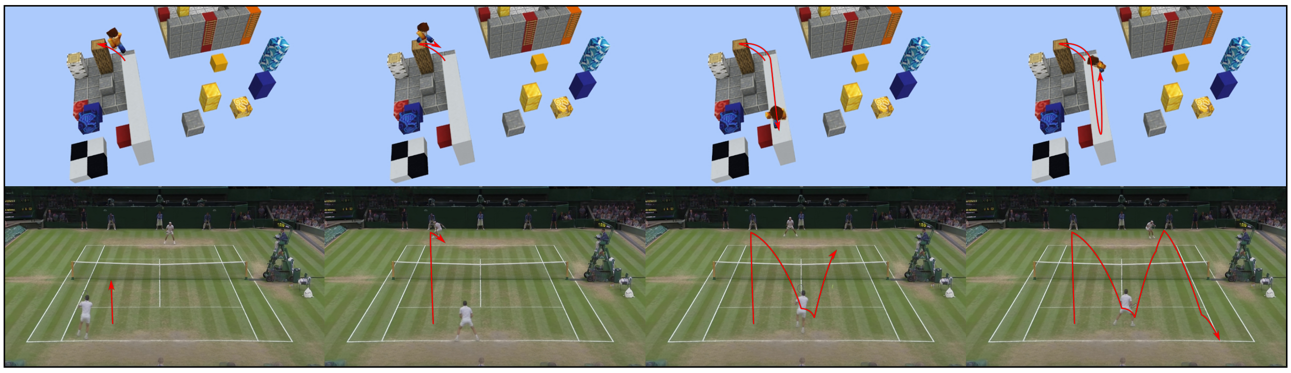


Figure 4.9: Sequences generated without any user conditioning signal. The actions of all players are controlled by the model that acts as the game AI. In tennis, the players produce a realistic exchange, with the bottom player advancing aggressively toward the net and the top player defeating him with a shot along the right sideline. The Minecraft player and tennis ball trajectories are highlighted for better visualization.

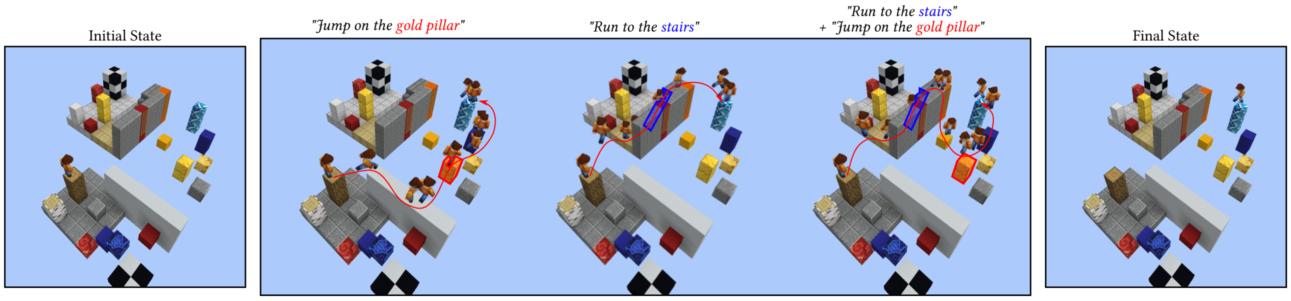

Initial State
"Jump on the gold pillar"
"Run to the stairs"
"Run to the stairs" + "Jump on the gold pillar"
Final State

Figure 4.10: 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.

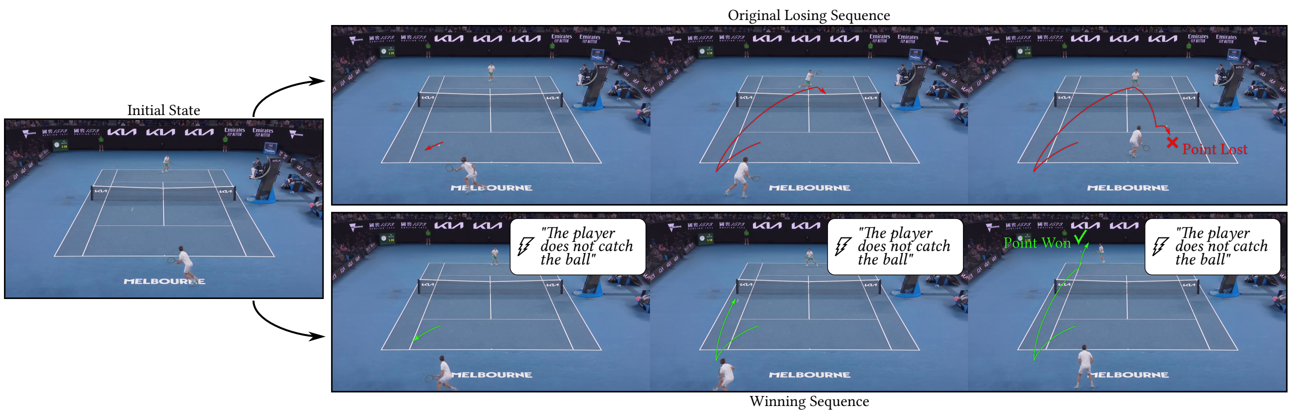

Original Losing Sequence
Initial State
Point Lost
"The player does not catch the ball"
"The player does not catch the ball"
Point Won
"The player does not catch the ball"
Winning Sequence

Figure 4.11: 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.

Our state representation is modular, where the style is one of the components. To demonstrate style swapping capabilities, in Fig. 4.5 we swap the style of the player $\boldsymbol{\omega}$ 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 PGM to perform novel view synthesis as shown in Fig. 4.6.

We now show a set of applications enabled by the animation model. In Fig. 4.7, 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 of the game's dynamics, the model can tackle more complex tasks such as modeling an opponent against which a user-controlled player can play (see Fig. 4.8), or even controlling all players in a scene without user intervention (see Fig. 4.9), 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 prompts consisting in 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. 4.10 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. 4.11 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.

4.5 Evaluation

In this section, we introduce our *Tennis* and *Minecraft* datasets (Sec. 4.5.1), describe our experimental protocol (Sec. 4.5.2), and perform evaluation of both the synthesis model (Sec. 4.5.5) and the animation model (Sec. 4.5.6).

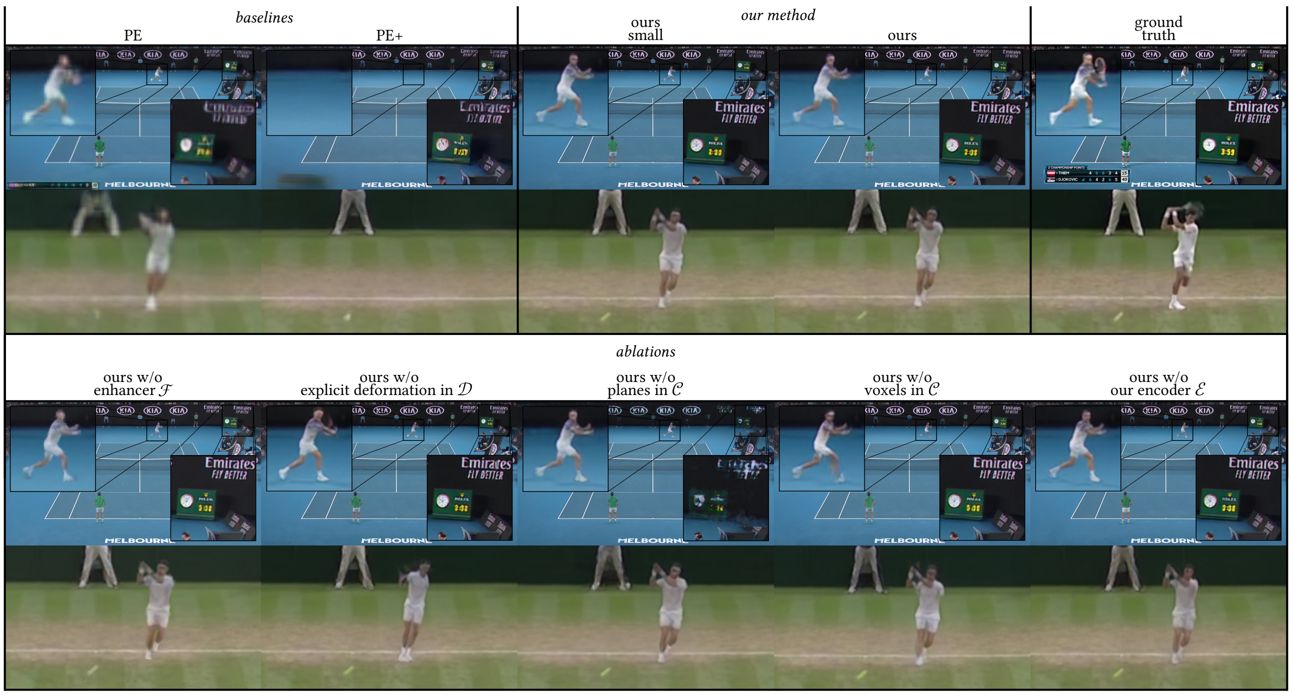

baselines
PE
PE+
our method
ours small
ours
ground truth
ablations
ours w/o enhancer $\mathcal{F}$
ours w/o explicit deformation in $\mathcal{D}$
ours w/o planes in $\mathcal{C}$
ours w/o voxels in $\mathcal{C}$
ours w/o our encoder $\mathcal{E}$

Figure 4.12: Synthesis model qualitative results on the *Tennis* dataset. Compared to PE [86], 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.

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.

Tennis dataset

We collect a dataset of broadcast tennis matches starting from the videos in [86]. 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 PGMs, we collect a wide range of annotations with a combination of manual and automatic methods:

- For each frame, we perform camera calibration.
- For each of the two players, we perform tracking and collect full SMPL [79] 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. 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 [5] or high-level action descriptions weakly aligned with video content [88], 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 [86] 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 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 [86].

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 [110]:

- 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.
- 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.

4.5.2 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* [154] is a standard metric for evaluating the reconstruction quality of the generated images
- *FID* [40] is a widely-used metric for image generation quality
- *FVD* [136] is a standard metric for assessing the quality of generated videos
- *Average Detection Distance (ADD)* [85] measures the average distance in pixels between the bounding box centers of ground truth bounding boxes and bounding boxes obtained from the generated sequences through a pretrained detector [109]

- *Missing Detection Rate (MDR)* [85] 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)* [30] 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 [86].
- *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.

4.5.3 Implementation and Training Details

Synthesis model architecture 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 [86].

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 [38]. 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 $\boldsymbol{\omega}$ of corresponding objects [86].

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

$H_W = W_W = D_W = 32$. For the *Minecraft* scene, the size of the voxel $\mathbf{V}$ is increased to $H_V = W_V = D_V = 128$. The *Minecraft* skybox is represented with feature planes $\mathbf{P}$ 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 $\mathbf{P}$ 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.

Animation model architecture For the text encoder, we model $\mathcal{T}_{\text{enc}}$ as a frozen T5-Large [106] model and $\mathcal{T}_{\text{agg}}$ as a transformer encoder [137] with 4 layers, 8 heads, and a feature size of 1024. For each sequence, the output $\mathbf{a}^{\text{emb}}$ 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 [114], we found alternative choices for $\mathcal{T}_{\text{enc}}$ (T5-Small, T5-Base [106] and the CLIP text encoder [105]) 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 [118] 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 ν using sinusoidal position encodings [137]. Additionally, to enable the model to better distinguish between noisy sequence entries and conditioning signals, we find it beneficial to condition also on $\mathbf{m^s}$ and $\mathbf{m^a}$ 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 [157]; 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.

Synthesis model training We employ a reduced learning rate of $1e-5$ for the 3D CNNs that model the canonical radiance field voxels $\mathbf{V}$ and blending weights $\mathbf{W}$ 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.

Animation model training We create masks $\mathbf{m^s}$ and $\mathbf{m^a}$ 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 $\mathbf{m^a} = 1$, 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.

4.5.4 Inference Details

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.* [87] 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.

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

The text encoder $\mathcal{T}$ produces text embeddings $\mathbf{a}^{\text{emb}}$ as in Eq. (4.7). Successively, the *reverse process* is started at diffusion time $k = K$, with $\mathbf{s}^p_K$ sampled from the normal distribution. The DDPM sampler [42] queries the temporal model according to Eq. (4.8) to progressively denoise $\mathbf{s}^p_k$ and obtain the predicted sequence $\mathbf{s}^p = \mathbf{s}^p_0$. The final sequence is obtained as $\mathbf{s} = \mathbf{s}^p + \mathbf{s}^c$, following Eq. (4.6).

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 ν_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 ν_2 and extend the sampled sequence $\mathbf{s}$ with new states between existing ones, that we call keyframes, until the sequence length corresponding to ν_2 is reached. This sequence constitutes the new $\mathbf{s}^c$. Any previous action conditioning is copied in a new $\mathbf{a}^c$ 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 $\mathbf{s}$ is produced at the final framerate ν_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.

4.5.5 Synthesis Model Evaluation

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

Comparison to Baselines

We evaluate our method against Playable Environments (PE) [86], 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 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. 4.1, while qualitative results are shown in Fig. 4.12. 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. 4.12, PE and PE+

Tennis	LPIPS↓	FID↓	FVD↓	ADD↓	MDR↓
PE† [86]	0.188	11.5	349	3.74	0.200
PE+ [86]	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 $\mathcal{C}$	0.241	30.4	1064	2.94	0.0611
w/o voxels in $\mathcal{C}$	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† [86]	0.0235	13.9	21.5	5.77	0.0412
PE+ [86]	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

Table 4.1: Comparison with baselines and ablation of the synthesis model. MDR in %, ADD in pixels. Note that FID and FVD are computed on images downscaled 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. 4.12). † denotes output in 512x288px rather than 1024x576px resolution.

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.

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 $\tilde{\mathbf{I}}$ as our output; we remove the explicit deformation modeling procedure in $\mathcal{D}$ of Sec. 4.3.1 and substitute it with an MLP directly predicting the deformation using a learnable pose code as in [86, 134]; we remove the plane-based canonical volume representation in $\mathcal{C}$ for planar objects and use an MLP instead; we remove the voxel-based volume representation in $\mathcal{C}$ and use an MLP instead; we substitute our style encoder $\mathcal{E}$ with an ad-hoc encoder for each object in the scene, following [86].

We perform the ablation on the *Tennis* dataset and show results in Tab. 4.1 and Fig. 4.12. 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. 4.12). 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. 4.12). 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 [86], we notice fewer details in scene elements.

4.5.6 Animation Model Evaluation

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

Comparison to Baselines

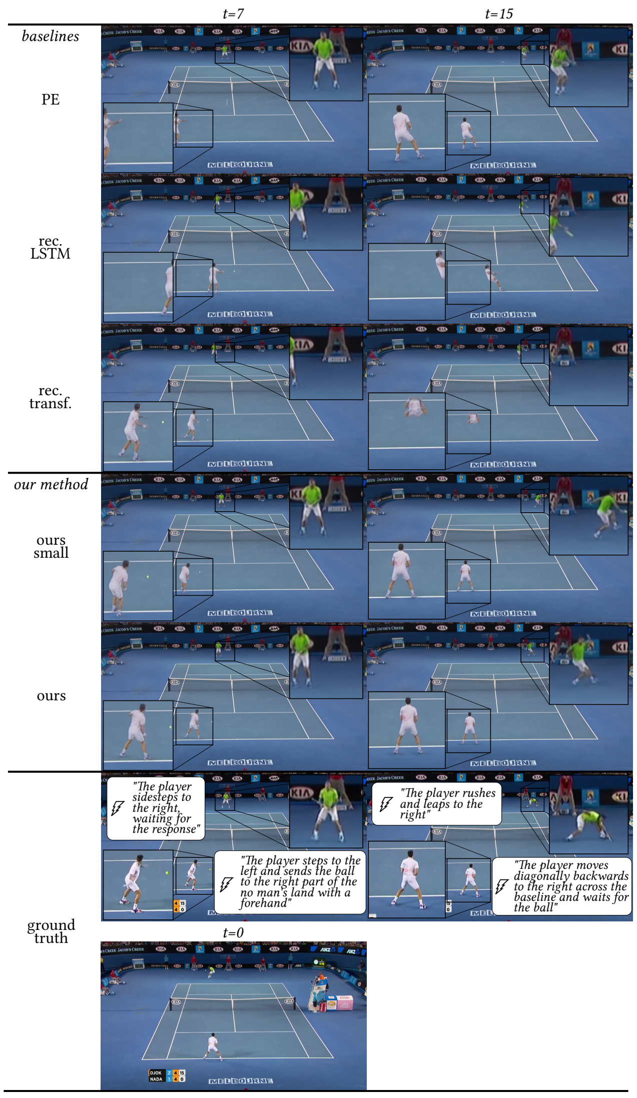
Similarly to the synthesis model, we compare our animation model against the one of Playable Environments (PE) [86], the most related to our work since it operates on a similar environment representation. While the baseline jointly learns discrete actions and generates sequences conditioned on such actions, 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 for all inference tasks in Tab. 4.2. Our method outperforms the baseline in all evaluation tasks according to both L2 and FD metrics. From the qualitative results in Fig. 4.13 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.

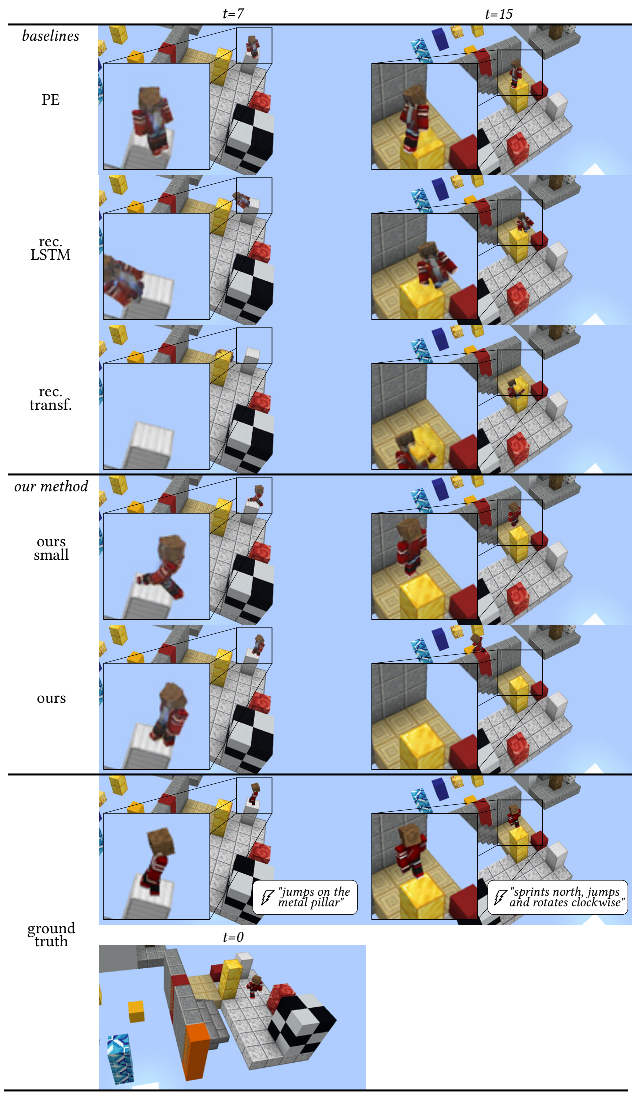
Ablation

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 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. 4.2. Our model consistently outperforms the baselines in terms of FD, showing a better ability to capture realistic sequences. Consistently with our assessment in Sec. 4.5.6, Fig. 4.13 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.


t=7
t=15
baselines
PE
rec. LSTM
rec. transf.
our method
ours small
ours
ground truth
"The player sidesteps to the right, waiting for the response"
"The player rushes and leaps to the right"
"The player steps to the left and sends the ball to the right part of the no man's land with a forehand"
"The player moves diagonally backwards to the right across the baseline and waits for the ball"
t=0

(a) *Tennis*



(b) *Minecraft*

Figure 4.13: Qualitatives results on the *Tennis* and *Minecraft* 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

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 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 the 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.

Animation Model Masking Strategies Ablation

In Tab. 4.3, we ablate the contribution of the animation model training masking strategies (see Sec. 4.5.3) on the Tennis dataset. We group the masking strategies in four groups of semantically-related strategies: "Random" (i + ii) which masks random sequence elements, "Block" (iii + iv) which masks blocks of contiguous timesteps, "Last" (v) which masks the last timesteps of the sequence, "Opponent" (vi) that masks a randomly chosen set of properties in the whole sequence.

The use of only the "Random" masking strategy results in the worst performance as the model only learns to interpolate between adjacent values and fails on tasks requiring long predictions. Adding the "Block" strategy enables the model to learn how to interpolate between values separated by larger gaps, producing good results in the sequence completion task. The addition of the "Last" training strategy enables the model to learn how to produce future states from a given initial one, improving the results on both video prediction tasks. The "Opponent" masking strategy enables the model to recover properties that are missing in the whole sequence, enabling best performance on the opponent modeling task. Finally, combining all masking strategies enables the model to jointly learn these capabilities, producing the best results.

Animation Model Dataset Size Ablation

In Tab. 4.4, we analyze the performance of the animation model as a function of the available portion of the training data on the Tennis dataset. The model performance gradually reduces as the amount of training data shrinks. When the amount of available training data falls below 60% of the original dataset size, the model overfits to the training data, yielding poor performance.

4.6 Conclusions

In this chapter, we demonstrate the feasibility of learning game models able to answer challenging user prompts. We introduce a two-stage framework for the creation of promptable game models. The first stage is comprised of a compositional NeRF representation that renders states of environments to frames. To render high-quality videos we build efficient representations of radiance fields that allow direct rendering of high-resolution frames without the need for upsampling networks. The second stage is responsible for learning the dynamics of the environment that evolve states in time. A masked diffusion transformer is introduced that generates sequences of states based on user-provided conditioning signals, or prompts, composed of desired states of the environment and textual descriptions of actions.

We show that textual action representations are critical for unlocking fine-grained control over the generation process, and enabling compelling constraint- and goal-driven generation applications. While the acquisition of these textual representations entails annotation efforts, the associated collections costs remains orders of magnitude smaller with respect to the costs associated with traditional game development and recent image and video captioning methods [74, 75, 150, 155, 158] are a promising avenue to automate this task. The framework is evaluated on a real-world tennis dataset and a synthetic minecraft dataset showcasing complex actions and agent behavior.

Our method obtains state-of-the-art results and allows the generation of videos given complex high-level goals, a capability not demonstrated in previous works. These results, jointly with two richly-annotated text-video datasets, pave the way towards learning promptable models for complex, real-world scenes.

	Tennis						*Minecraft*					
	Position		*Root angle*		*Joints 3D*		*Position*		*Root angle*		*Joints 3D*	
	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓
Action conditioned video prediction												
PE	3.117	87.688	1.182	12.627	0.277	30.711	2.720	90.904	1.822	23.949	0.365	47.956
Rec. LSTM	1.753	7.413	1.100	8.416	0.234	18.455	2.623	54.927	2.040	62.363	0.579	118.592
Rec. Transf.	1.183	2.996	0.913	7.566	0.212	15.976	2.798	76.582	1.794	52.677	0.506	100.731
Ours Small	1.244	1.071	1.187	0.601	0.178	1.570	0.533	2.494	0.901	5.624	0.145	4.083
Ours	1.064	0.846	0.961	0.421	0.153	1.049	0.523	2.582	0.749	4.578	0.135	3.794
Unconditional video prediction												
PE	3.973	146.019	1.604	30.448	0.437	78.835	3.994	197.434	2.111	59.112	0.370	46.665
Rec. LSTM	2.064	11.283	1.224	14.860	0.264	28.736	2.850	68.915	1.999	69.886	0.581	121.187
Rec. Transf.	1.649	10.514	1.123	15.648	0.251	27.258	2.834	76.780	1.795	50.871	0.480	87.584
Ours Small	2.352	2.271	1.455	0.781	0.213	1.827	2.341	9.795	1.814	8.969	0.199	4.422
Ours	1.925	1.377	1.277	0.518	0.192	1.261	2.330	11.032	1.685	5.648	0.197	4.385
Sequence completion												
PE	4.353	641.976	0.903	13.955	0.251	62.981	1.504	29.582	0.926	10.634	0.197	24.096
Rec. LSTM	1.581	5.507	0.697	2.517	0.143	10.443	1.401	18.044	1.066	17.664	0.309	59.750
Rec. Transf.	1.169	3.735	0.631	2.514	0.138	10.519	0.830	6.232	0.700	4.822	0.170	21.615
Ours Small	1.578	2.243	0.832	0.560	0.114	0.851	0.379	1.095	0.516	3.456	0.077	2.265
Ours	1.153	1.349	0.703	0.288	0.101	0.558	0.343	0.830	0.433	2.022	0.065	1.899
Opponent modeling												
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						
Average												
PE	3.291	229.112	1.126	15.953	0.303	53.242	2.739	105.973	1.620	31.232	0.311	39.572
Rec. LSTM	1.597	7.253	0.907	7.051	0.193	16.735	2.292	47.296	1.702	49.971	0.489	99.843
Rec. Transf.	1.074	4.402	0.767	6.838	0.175	14.845	2.154	53.198	1.430	36.123	0.385	69.977
Ours Small	1.380	1.443	1.014	0.560	0.148	1.253	1.084	4.461	1.077	6.016	0.140	3.590
Ours	1.099	0.929	0.844	0.356	0.129	0.836	1.065	4.815	0.956	4.083	0.132	3.360

Table 4.2: Animation model comparison with baselines and ablation on the Tennis and Minecraft datasets. Position and Joints 3D in meters, Root angle in axis-angle representation.

Table 4.3: Ablation on the Tennis dataset of different training masking strategies of Sec. 4.5.3: “Random” (i + ii), “Block” (iii + iv), “Last” (v), “Opponent” (vi). Position and Joints 3D in meters, Root angle in axis-angle representation.

	Position		*Root angle*		*Joints 3D*	
	L2↓	FD↓	L2↓	FD↓	L2↓	FD↓
	Action conditioned video prediction					
Random	3.350	78.825	1.703	5.455	0.241	6.242
Random + Block	1.814	1.906	1.458	1.069	0.203	2.033
Random + Last	1.335	1.005	1.241	0.783	0.186	1.764
Random + Opponent	1.355	1.279	1.225	0.687	0.185	1.681
All	1.244	1.071	1.187	0.601	0.178	1.570
	Unconditional video prediction					
Random	4.583	227.295	1.663	5.814	0.287	12.649
Random + Block	2.621	3.995	1.569	0.988	0.221	2.371
Random + Last	2.229	2.296	1.503	0.960	0.215	2.099
Random + Opponent	3.358	13.365	1.661	0.992	0.238	2.364
All	2.352	2.271	1.455	0.781	0.213	1.827
	Opponent modeling					
Random	2.246	12.463	0.882	1.040	0.125	1.411
Random + Block	2.031	4.768	0.901	0.333	0.123	0.934
Random + Last	1.965	4.623	0.861	0.631	0.125	1.143
Random + Opponent	1.486	1.191	0.823	0.359	0.115	0.755
All	1.578	2.243	0.832	0.560	0.114	0.851
	Sequence completion					
Random	0.581	2.030	0.670	0.503	0.104	1.199
Random + Block	0.364	0.223	0.590	0.331	0.090	0.825
Random + Last	0.414	0.387	0.614	0.350	0.093	0.904
Random + Opponent	0.494	0.608	0.659	0.403	0.096	0.896
All	0.344	0.187	0.581	0.301	0.088	0.765
	Average					
Random	2.690	80.153	1.229	3.203	0.189	5.375
Random + Block	1.707	2.723	1.129	0.680	0.159	1.541
Random + Last	1.486	2.078	1.055	0.681	0.155	1.477
Random + Opponent	1.673	4.111	1.092	0.610	0.158	1.424
All	1.380	1.443	1.014	0.560	0.148	1.253

Table 4.4: Animation model performance as a function of the dataset size 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↓
	Action conditioned video prediction					
20%	4.237	425.180	1.645	10.917	0.251	14.293
40%	3.723	244.390	1.543	6.925	0.236	9.027
60%	1.331	1.403	1.236	0.954	0.189	1.869
80%	1.283	1.024	1.207	0.714	0.183	1.607
100%	1.244	1.071	1.187	0.601	0.178	1.570
	Unconditional video prediction					
20%	4.391	445.758	1.736	12.150	0.260	15.167
40%	4.287	323.800	1.701	6.922	0.254	10.196
60%	2.446	4.059	1.537	1.216	0.222	2.257
80%	2.402	2.475	1.510	0.955	0.216	1.892
100%	2.352	2.271	1.455	0.781	0.213	1.827
	Opponent modeling					
20%	3.829	321.502	0.993	2.212	0.130	2.469
40%	3.086	170.115	0.937	1.771	0.129	1.979
60%	1.673	2.094	0.837	0.416	0.119	0.862
80%	1.550	1.606	0.817	0.450	0.114	0.749
100%	1.578	2.243	0.832	0.560	0.114	0.851
	Sequence completion					
20%	1.456	71.621	0.738	2.809	0.118	3.492
40%	1.184	37.588	0.706	1.808	0.111	2.633
60%	0.370	0.228	0.608	0.291	0.094	0.853
80%	0.372	0.237	0.589	0.231	0.090	0.773
100%	0.344	0.187	0.581	0.301	0.088	0.765
	Average					
20%	3.478	316.015	1.278	7.022	0.190	8.855
40%	3.070	193.973	1.222	4.357	0.183	5.959
60%	1.455	1.946	1.054	0.719	0.156	1.460
80%	1.402	1.335	1.031	0.588	0.151	1.255
100%	1.380	1.443	1.014	0.560	0.148	1.253

Chapter 5

Discussion

In this chapter, we conclude this doctoral thesis by highlighting applications of the presented methodologies (see Sec. 5.1), ethical considerations regarding the proposed generative frameworks (see Sec. 5.2), discussing their limitations (see Sec. 5.3), highlighting future research directions (see Sec. 5.4) and giving final remarks (see Sec. 5.5).

5.1 Applications

In this thesis, we discuss a range of techniques that enable the creation of playable experiences from a set of annotated videos. We note, however, that the applicability of such techniques spans several areas beyond videogame creation.

Thanks to the fine grained control on the generation process, the presented techniques are amenable to use as video editing methods for complex scenes, enabling changes in movement, action, style and identities of its characters. Such capabilities find relevance in diverse domains, including the tactical analysis of sports matches. In this context, experts engage in match analysis by superimposing player movement patterns and shot trajectories onto displayed video frames. This visual representation aims to showcase potential alternative courses of action to the audience. The proposed techniques offer an additional dimension to this process, allowing the expert to simulate the proposed course of action and generate an associated video. This not only validates the proposed analysis but also enhances the engagement for the audience.

The animation module presented in Chapter 4 learns the dynamics complex environments with multiple interacting agents. Most interestingly, not only it enables its simulation from a given initial state, but enables the explanation of observed outcomes, such as deriving possible ways in which the modeled system can end in a given state. These capabilities open up venues of application in the realm of simulation, helping designers understanding the dynamics of their modeled systems.

5.2 Ethical Considerations

The techniques described in this doctoral thesis belong to the realm of video editing methodologies. These methodologies, along with those falling under the broader category of generative AI, are responsible for a paradigm shift in the landscape of multimedia content creation. They have rendered it feasible for individuals lacking professional expertise in multimedia to create sophisticated special effects and attain realistic alterations in multimedia content. This democratization of multimedia production empowers ordinary users with formidable tools to manifest their creativity. However, it is imperative to recognize that these transformative capabilities may also be harnessed for nafarious purposes, such as the manipulation of video content to deceive or disseminate malicious messages.

The methodologies outlined in this thesis possess inherent attributes that provide safeguards against their malevolent applications. The proposed CADDY framework for Playable Video Generation operates on scenes containing a single agent and no camera movement. Furthermore, it necessitates the utilization of a comprehensive dataset composed of multiple videos for training. This requirements restrict the applicability of the method, rendering it unsuitable for the manipulation of scenes with limited source footage. Additionally, the system provides users with a finite set of discrete actions, which, in turn, renders it challenging for them to achieve a level of control sufficient to perpetrate deception.

Conversely, the design of Playable Environments and Promptable Game Models permits sophisticated control over the scene contents and visuals, but assumes the availability of multiple camera-calibrated observations of a single scene in the training dataset. Furthermore, it requires that the desired edit has been observed at least once in the training data. These prerequisites serve as a safeguard against the illicit application of the method to tamper a single video, where the volume of available data would be inadequate and the desired alterations unlikely to have been previously depicted.

In view of the rapid and pervasive proliferation of generative AI techniques, it is of primary importance to foster public awareness and education about such generative techniques. Moreover, the development of automated techniques for the detection of artificially generated content constitutes an important line of defense against the risks associated with malicious or deceptive contents.

5.3 Limitations

Within the scope of this doctoral thesis, we have presented three methodologies that facilitate the creation of interactive game-like experiences, each of which introduces additional dimensions of control and seeks to address shortcomings present in preceding approaches. Notwithstanding the merits of our most recent work, Promptable Game Models, which showcases the potential for generating immersive interactive experiences with a granular level of control and embedded "game AI" capabilities, certain aspects of this framework continue to pose a limitation.

The steady increase in output resolution, which has quadrupled since the inception of Playable Video Generation, in conjunction with the incorporation of 3D scene representations, has led to a significant escalation in computational demands, despite careful model optimization. Ensuring the possibility of real-time performance is an important aspect of playability, and the ongoing enhancement of neural radiance fields and the refinement of diffusion model distillation techniques present promising avenues to enhance the overall performance of the model.

Our proposed frameworks are constrained in their ability to generate actions that lie substantially outside the bounds of the training data distribution. For instance, they are incapable of generating actions as executing a backflip or performing a push-up, which have not been previously demonstrated in the training dataset. Consequently, the creative applications of these frameworks are limited to interpolations between known concepts. Moreover, since the models are conditioned on datasets exclusively containing plausible actions, they exhibit undefined behavior when confronted with actions that are *implausible*. This includes scenarios such as attempting to strike a ball while moving in the opposite direction of the intercept path or leaping onto a pillar that lies to far for a plausible jump. In such instances, our observations reveal that the model typically neglects the implausible components of the command, generating instead the nearest plausible alternative command or, on less frequent occasions, manifesting implausible outcomes, such as improbable long-distance jumps.

Playable Environments and Promptable Game Models assume a static scene geometry, thus precluding the generation of video content in unseen environments. This inherent constraint restricts the applicability of these models to situations in which multiple video sources are available, collectively depicting a scene with unchanging geometry.

Lastly, each of our proposed frameworks has escalated the need for annotation. While Playable Video Generation operates within a completely unsupervised framework, Playable Environments introduces the prerequisites of camera calibration and localization of objects within the scene. Furthermore, Promptable Game Models necessitate textual action annotations. Although the availability of such annotations has markedly enriched the capabilities of our models, their acquisition poses a challenge to the widespread adoption of our methodology across diverse environments. To mitigate this issue, there is potential to employ automated methods for generating various types of annotations, with the most challenging being the provision of textual action annotations. Notably, the recent advancements in image and video captioning techniques [74, 75, 156, 159] offer promise in the creation of automated dataset captioning pipelines.

5.4 Future Work

This thesis has delved into methodologies for the generation of interactive experiences resembling video games. A common thread that unites the various proposed methods is the search for fine-grained control within the video generation process and precision in simulating 3D environments—attributes that fundamentally underpin playability. Notably, the video generation techniques presented in this work are tailored to a single environment.

In contrast, recent advancements in diffusion-based video generation methods [6, 41, 123] have showcased an impressive capacity to generate video content spanning a wide spectrum of scenes via text conditioning. However, these approaches often lack the granularity of control and the precision required to accurately simulate the environment.

Bringing together the capability to control video generation in a playable manner and the ability to recreate diverse 3D environments within a framework capable of generating an extensive array of video content represents an promising direction that could enable a range of significant applications such as the fine-grained editing of real-world footage. Notably, recently proposed methods for constructing radiance fields from real-world videos without the need for camera calibration [124], and the advent of automated video captioning techniques [74, 75, 156, 159], hold promise in establishing a large-scale dataset of videos enriched with 3D and textual annotations. This dataset could serve as the foundational resource for training models that can integrate these capabilities—a promising avenue for future research.

5.5 Conclusions

This doctoral thesis has been dedicated to the introduction and exploration of methodologies for the development of playable video generation models. The novel proposed video generation approaches are designed to learn how to simulate environments depicted in observed datasets, and the actions of their agents. In doing so, these models allow the generation of videos with a high degree of user control, akin to the interaction experienced by users in the realm of video games.

The development of such techniques begins with the formulation of Playable Video Generation, a novel formulation that seeks to address the problem in a completely unsupervised manner. This problem consists in learning a video generator capable of being manipulated through a set of discrete actions, using an unlabeled dataset of videos as its foundation. To this end, we introduce a novel framework for playable video generation named Clustering for Action Decomposition and Discovery (CADDY). This framework relies on a novel action representation that captures both high-level discrete action components and low-level continuous action variations. The efficacy of CADDY is demonstrated on diverse datasets, encompassing real-world tennis and robotics data, as well as synthetic Atari video game datasets. In each instance, our proposed framework outperforms the state-of-the-art, yielding high-quality videos.

Recognizing the inherently three-dimensional nature of the environments modeled by CADDY and the limitations of representing camera movement within two-dimensional frameworks, we advance our exploration to Playable Environments. This novel concept extends the foundational principles of Playable Video Generation into the three-dimensional realm. Within this framework, we introduce a methodology for crafting playable environments that accommodate multiple deformable agents. Users are given granular control over the agents, the stylistic attributes of objects within the environment, and explicit control over the camera perspective. Our framework relies on a compositional neural radiance field formulation that independently models each object in the scene, enabling the rendering of the environment state from a user-controlled viewpoint. To establish a benchmark for future research, we introduce two datasets, featuring a synthetic Minecraft dataset and a real-world tennis dataset annotated with camera-calibrated videos and 2D object bounding boxes. Our results show that the proposed method surpasses the previous state-of-the-art.

In the final chapter, we introduce Promptable Game Models, which integrate natural language as a means of enabling fine-grained control over the video generation process. The model also demonstrates an ability to simulate intelligent agents that respond to natural language instructions and exhibit intelligent behaviors consistent with their own “game AI”. This novel framework leverages a masked diffusion transformer, enabling non-autoregressive generation of sequences of environment states that fulfill a set of user-defined conditions expressed through prompts, which encompass both natural language actions for the agents and desired environment states. Moreover, we introduce efficient neural radiance field representations that enable the rendering of high-resolution videos without the need for upsampling, a development crucial for overcoming checkerboard artifacts and enhancing the modeling of smaller objects, critical aspects of existing methods. To foster future research in this domain, we release two datasets enriched with camera calibration data, 3D object poses, and textual action annotations. Our results demonstrate the superiority of our proposed framework in terms of visual fidelity and the capabilities it affords in generating scene states, outperforming the current state-of-the-art methodologies.

Bibliography

[1] Minecraft. `https://www.minecraft.net`. Accessed: 2021-11-12.

[2] Panos Achlioptas, Ian Huang, Minhyuk Sung, Sergey Tulyakov, and Leonidas Guibas. ChangeIt3D: Language-assisted 3d shape edits and deformations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[3] Nikos Athanasiou, Mathis Petrovich, Michael J. Black, and Gül Varol. Teach: Temporal action compositions for 3d humans. In *International Conference on 3D Vision (3DV)*, September 2022.

[4] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. In *International Conference on Learning Representations (ICLR)*, 2018.

[5] Max Bain, Arsha Nagrani, Gül Varol, and Andrew Zisserman. 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)*, 2021.

[6] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, 2016.

[8] Michael Büttner and Simon Clavet. Motion matching - the road to next gen animation. In *In Proc. of Nucl.ai 2015*, 2015.

[9] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference of Computer Vision (ECCV)*, 2018.

[10] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[11] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[12] Gaurav Chaurasia, Sylvain Duchêne, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics*, 2013.

[13] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022.

[14] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations (ICLR)*, 2021.

[15] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[16] Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *CoRR*, 2017.

[17] Aidan Clark, Jeff Donahue, and Karen Simonyan. Efficient video generation on complex datasets. *CoRR*, 2019.

[18] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 1984.

[19] Cassidy Curtis, Sigurdur Orn Adalgeirsson, Horia Stefan Ciurdar, Peter McDermott, JD Velásquez, W. Bradley Knox, Alonso Martinez, Dei Gaztelumendi, Norberto Adrian Goussies, Tianyu Liu, and Palash Nandy. Toward believable acting for autonomous animated characters. In *Proceedings of the 15th ACM SIGGRAPH Conference on Motion, Interaction and Games*, 2022.

[20] Rishabh Dabral, Muhammad Hamza Mughal, Vladislav Golyanik, and Christian Theobalt. Mofusion: A framework for denoising-diffusion-based motion synthesis. *arXiv*, 2022.

[21] Aram Davtyan and Paolo Favaro. Controllable video generation through global and local motion dynamics. In *Proceedings of the European Conference of Computer Vision (ECCV)*, 2022.

[22] Sander Dieleman, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H. Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, Curtis Hawthorne, Rémi Leblond, Will Grathwohl, and Jonas Adler. Continuous diffusion for categorical data. *arXiv*, 2022.

[23] Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. Temporal cycle-consistency learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[24] Frederik Ebert, Chelsea Finn, Alex X. Lee, and S. Levine. Self-supervised visual planning with temporal skip connections. In *CoRL*, 2017.

[25] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[26] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 1971.

[27] Joseph L Fleiss, Bruce Levin, Myunghee Cho Paik, et al. The measurement of interrater agreement. *Statistical methods for rates and proportions*, 1981.

[28] Vincent Fortuin, Dmitry Baranchuk, Gunnar Rätsch, and Stephan Mandt. Gp-vae: Deep probabilistic time series imputation. In *International conference on artificial intelligence and statistics*. PMLR, 2020.

[29] Jean-Yves Franceschi, Edouard Delasalles, Mickael Chen, Sylvain Lamprier, and P. Gallinari. Stochastic latent residual video prediction. *ArXiv*, 2020.

[30] Maurice Fréchet. Sur la distance de deux lois de probabilité. *Comptes Rendus Hebdomadaires des Seances de L Academie des Sciences*, 1957.

[31] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[32] Tsu-Jui Fu, Licheng Yu, Ning Zhang, Cheng-Yang Fu, Jong-Chyi Su, William Yang Wang, and Sean Bell. Tell me what happened: Unifying text-guided video completion via multimodal masked video generation. *arXiv*, 2022.

[33] Oran Gafni, Lior Wolf, and Yaniv Taigman. Vid2game: Controllable characters extracted from real-world videos. In *International Conference on Learning Representations (ICLR)*, 2020.

[34] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[35] Jason Gregory. *Game Engine Architecture*. CRC Press, 2018.

[36] Ligong Han, Jian Ren, Hsin-Ying Lee, Francesco Barbieri, Kyle Olszewski, Shervin Minaee, Dimitris Metaxas, and Sergey Tulyakov. Show me what and tell me how: Video synthesis via multimodal conditioning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[37] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. GANcraft: Unsupervised 3D Neural Rendering of Minecraft Worlds. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[39] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

[40] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[41] 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. Imagen video: High definition video generation with diffusion models. *arXiv*, 2022.

[42] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[43] Jonathan Ho, Tim Salimans, Alexey A. Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In *ICLR Workshop on Deep Generative Models for Highly Structured Data*, 2022.

[44] Daniel Holden, Oussama Kanoun, Maksym Perepichka, and Tiberiu Popa. Learned motion matching. *ACM Transactions on Graphics (TOG)*, 2020.

[45] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, 2017.

[46] Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, and Jie Tang. Cogvideo: Large-scale pretraining for text-to-video generation via transformers. 2022.

[47] Jiahui Huang, Yuhe Jin, Kwang Moo Yi, and Leonid Sigal. Layered controllable video generation. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *"Proceedings of the European Conference of Computer Vision (ECCV)"*, 2022.

[48] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[49] Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

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

[51] Xu Ji, Andrea Vedaldi, and João F. Henriques. Invariant information clustering for unsupervised image classification and segmentation. *In Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[52] Zhang Jiakai, Liu Xinhang, Ye Xinyi, Zhao Fuqiang, Zhang Yanshun, Wu Minye, Zhang Yingliang, Xu Lan, and Yu Jingyi. Editable free-viewpoint video using a layered neural representation. In *SIGGRAPH*, 2021.

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

[54] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[55] Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. Drivegan: Towards a controllable high-quality neural simulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[56] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. 2020.

[57] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to Simulate Dynamic Environments with GameGAN. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[58] Yunji Kim, Seonghyeon Nam, In Cho, and Seon Joo Kim. Unsupervised keypoint learning for guiding class-conditional video prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[59] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, 2015.

[60] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *stat*, 2014.

[61] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations (ICLR)*, 2020.

[62] Juil Koo, Ian Huang, Panos Achlioptas, Leonidas J Guibas, and Minhyuk Sung. Partglot: Learning shape part segmentation from language reference games. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[63] Johannes Kopf, Fabian Langguth, Daniel Scharstein, Richard Szeliski, and Michael Goesele. Image-based rendering in the gradient domain. *ACM Transactions on Graphics*, 2013.

[64] Zhengfei Kuang, Kyle Olszewski, Menglei Chai, Zeng Huang, Panos Achlioptas, and Sergey Tulyakov. Neroic: Neural rendering of objects from online image collections. *ACM Transactions on Graphics (TOG)*, 2022.

[65] Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A conditional flow-based model for stochastic video generation. In *International Conference on Learning Representations*, 2019.

[66] Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A conditional flow-based model for stochastic video generation. In *International Conference on Learning Representations (ICLR)*, 2020.

[67] Abhijit Kundu, Kyle Genova, Xiaoqi Yin, Alireza Fathi, Caroline Pantofaru, Leonidas J. Guibas, Andrea Tagliasacchi, Frank Dellaert, and Thomas A. Funkhouser. Panoptic neural fields: A semantic object-aware neural scene representation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[68] Yong-Hoon Kwon and Min-Gyu Park. Predicting future frames using retrospective cycle gan. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[69] Max W. Y. Lam, Jun Wang, Dan Su, and Dong Yu. BDDM: Bilateral denoising diffusion models for fast and high-quality speech synthesis. In *International Conference on Learning Representations (ICLR)*, 2022.

[70] Alex X. Lee, Richard Zhang, Frederik Ebert, P. Abbeel, Chelsea Finn, and S. Levine. Stochastic adversarial video prediction. *ArXiv*, 2018.

[71] Kyungho Lee, Seyoung Lee, and Jehee Lee. Interactive character animation by learning multi-objective control. *ACM Transactions on Graphics (TOG)*, 2018.

[72] Yichong Leng, Zehua Chen, Junliang Guo, Haohe Liu, Jiawei Chen, Xu Tan, Danilo Mandic, Lei He, Xiangyang Li, Tao Qin, sheng zhao, and Tie-Yan Liu. Binauralgrad: A two-stage conditional diffusion probabilistic model for binaural audio synthesis. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[73] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. 2000.

[74] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023.

[75] KunChang Li, Yinan He, Yi Wang, Yizhuo Li, Wenhai Wang, Ping Luo, Yali Wang, Limin Wang, and Yu Qiao. Videochat: Chat-centric video understanding, 2023.

[76] Ruilong Li, Julian Tanke, Minh Vo, Michael Zollhofer, Jurgen Gall, Angjoo Kanazawa, and Christoph Lassner. Tava: Template-free animatable volumetric actors. In *Proceedings of the European Conference of Computer Vision (ECCV)*, 2022.

[77] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. *arXiv*, 2022.

[78] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Ro{bert}a: A robustly optimized {bert} pretraining approach. In *International Conference on Learning Representations (ICLR)*, 2020.

[79] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. Smpl: A skinned multi-person linear model. *ACM Transactions on Graphics (TOG)*, 2015.

[80] Pauline Luc, A. Clark, S. Dieleman, D. Casas, Yotam Doron, Albin Cassirer, and K. Simonyan. Transformation-based adversarial video prediction on large-scale data. *ArXiv*, 2020.

[81] Zelun Luo, Boya Peng, De-An Huang, Alexandre Alahi, and Li Fei-Fei. Unsupervised learning of long-term motion dynamics for videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[82] Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: Adaptive coordinate networks for neural scene representation. *ACM Trans. Graph.*, 2021.

[83] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv*, 2015.

[84] Willi Menapace, Stéphane Lathuilière, and Elisa Ricci. Learning to cluster under domain shift. In *Proceedings of the European Conference of Computer Vision (ECCV)*, 2020.

[85] Willi Menapace, Stephane Lathuiliere, Sergey Tulyakov, Aliaksandr Siarohin, and Elisa Ricci. Playable video generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[86] Willi Menapace, Stéphane Lathuilière, Aliaksandr Siarohin, Christian Theobalt, Sergey Tulyakov, Vladislav Golyanik, and Elisa Ricci. Playable environments: Video manipulation in space and time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[87] Chenlin Meng, Ruiqi Gao, Diederik P Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *NeurIPS 2022 Workshop on Score-Based Methods*, 2022.

[88] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. HowTo100M: Learning a Text-Video Embedding by Watching Hundred Million Narrated Video Clips. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

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

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

[91] Norman Müller, Andrea Simonelli, Lorenzo Porzi, Samuel Rota Bulò, Matthias Nießner, and Peter Kontschieder. Autorf: Learning 3d object radiance fields from single view observations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[92] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)*, 2022.

[93] Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. Neural control variates. *ACM Trans. Graph.*, 2020.

[94] Michael Niemeyer and Andreas Geiger. CAMPARI: camera-aware decomposed generative neural radiance fields. In *In Proceedings of the International Conference on 3D Vision (3DV)*, 2021.

[95] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[96] Evangelos Ntavelis, Aliaksandr Siarohin, Kyle Olszewski, Chaoyang Wang, Luc Van Gool, and Sergey Tulyakov. Autodecoding latent 3d diffusion models, 2023.

[97] Manuel Serra Nunes, Atabak Dehban, Plinio Moreno, and José Santos-Victor. Action-conditioned benchmarking of robotic video prediction models: a comparative study. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[98] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[99] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

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

[101] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Transactions on Graphics (TOG)*, 2021.

[102] B. Peng, J. Lei, H. Fu, C. Zhang, T. Chua, and X. Li. Unsupervised video action clustering via motion-scene interaction constraint. *IEEE Transactions on Circuits and Systems for Video Technology*, 2020.

[103] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. 2017.

[104] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[105] 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. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021.

[106] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 2022.

[107] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *ArXiv*, 2022.

[108] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *Proceedings of the 38th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2021.

[109] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[110] ReplayMod. Replaymod. `https://github.com/ReplayMod/ReplayMod`, 2022. Accessed: 2021-11-12.

[111] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *arXiv*, 2021.

[112] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*.

[113] Oleh Rybkin, Karl Pertsch, Konstantinos G Derpanis, Kostas Daniilidis, and Andrew Jaegle. Learning what you can do before doing anything. In *International Conference on Learning Representations (ICLR)*, 2018.

[114] 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. Photorealistic text-to-image diffusion models with deep language understanding. *ArXiv*, 2022.

[115] Masaki Saito, Shunta Saito, Masanori Koyama, and Sosuke Kobayashi. Train sparsely, generate densely: Memory-efficient unsupervised training of high-resolution temporal gan. *International Journal of Computer Vision (IJCV).*

[116] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade W Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa R Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5b: An open large-scale dataset for training next generation image-text models. In *Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2022.

[117] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[118] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 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)*, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[119] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2015.

[120] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. Animating arbitrary objects via deep motion transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[121] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. First order motion model for image animation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[122] Aliaksandr Siarohin, Oliver Woodford, Jian Ren, Menglei Chai, and Sergey Tulyakov. Motion representations for articulated animation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[123] 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. Make-a-video: Text-to-video generation without text-video data, 2022.

[124] Cameron Smith, Yilun Du, Ayush Tewari, and Vincent Sitzmann. Flowcam: Training generalizable 3d radiance fields without camera poses via pixel-aligned scene flow. *arXiv*, 2023.

[125] Khurram Soomro and Mubarak Shah. Unsupervised action discovery and localization in videos. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[126] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 2014.

[127] Matt Stanton, Sascha Geddert, Adrian Blumer, Paul Hormis, Andy Nealen, Seth Cooper, and Adrien Treuille. Large-Scale Finite State Game Engines. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2016.

[128] Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. Neural state machine for character-scene interactions. *ACM Transactions on Graphics (TOG)*, 2019.

[129] Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics (TOG)*, 2020.

[130] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[131] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[132] Guy Tevet, Brian Gordon, Amir Hertz, Amit H. Bermano, and Daniel Cohen-Or. Motionclip: Exposing human motion generation to CLIP space. In Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Proceedings of the European Conference of Computer Vision (ECCV)*, Lecture Notes in Computer Science, 2022.

[133] Yu Tian, Jian Ren, Menglei Chai, Kyle Olszewski, Xi Peng, Dimitris N Metaxas, and Sergey Tulyakov. A good image generator is what you need for high-resolution video synthesis. In *International Conference on Learning Representations (ICLR)*, 2021.

[134] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. 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)*, 2021.

[135] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[136] Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv*, 2018.

[137] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 2017.

[138] Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. Phenaki: Variable length video generation from open domain textual description, 2022.

[139] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating the future by watching unlabeled video. *arXiv*, 2015.

[140] Yaohui Wand, Piotr Bilinski, Francois Bremond, and Antitza Dantcheva. Imaginator: Conditional spatio-temporal gan for video generation. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[141] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[142] Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. Scaling autoregressive video models. In *International Conference on Learning Representations (ICLR)*, 2020.

[143] Chung-Yi Weng, Brian Curless, Pratul P. Srinivasan, Jonathan T. Barron, and Ira Kemelmacher-Shlizerman. HumanNeRF: Free-viewpoint rendering of moving people from monocular video. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[144] Tianhao Walter Wu, Fangcheng Zhong, Andrea Tagliasacchi, Forrester Cole, and Cengiz Oztireli. D^2neRF: Self-supervised decoupling of dynamic and static objects from a monocular video. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[145] Ruijia Xu, Ziliang Chen, Wangmeng Zuo, Junjie Yan, and Liang Lin. Deep cocktail network: Multi-source unsupervised domain adaptation with category shift. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[146] Yinghao Xu, Menglei Chai, Zifan Shi, Sida Peng, Skorokhodov Ivan, Siarohin Aliaksandr, Ceyuan Yang, Yujun Shen, Hsin-Ying Lee, Bolei Zhou, and Tulyakov Sergy. Discoscene: Spatially disentangled generative radiance field for controllable 3d-aware scene synthesis. *arxiv*, 2022.

[147] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2021.

[148] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, Ben Hutchinson, Wei Han, Zarana Parekh, Xin Li, Han Zhang, Jason Baldridge, and Yonghui Wu. Scaling autoregressive models for content-rich text-to-image generation, 2022.

[149] Wentao Yuan, Zhaoyang Lv, Tanner Schmidt, and Steven Lovegrove. Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[150] Hang Zhang, Xin Li, and Lidong Bing. Video-llama: An instruction-tuned audio-visual language model for video understanding, 2023.

[151] Haotian Zhang, Cristobal Sciutto, Maneesh Agrawala, and Kayvon Fatahalian. Vid2player: Controllable video sprites that behave and appear like professional tennis players. *arXiv*, 2020.

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

[153] Mingyuan Zhang, Zhongang Cai, Liang Pan, Fangzhou Hong, Xinying Guo, Lei Yang, and Ziwei Liu. Motiondiffuse: Text-driven human motion generation with diffusion model. *arXiv*, 2022.

[154] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[155] Yue Zhao, Ishan Misra, Philipp Krähenbühl, and Rohit Girdhar. Learning video representations from large language models, 2022.

[156] Yue Zhao, Ishan Misra, Philipp Krähenbühl, and Rohit Girdhar. Learning video representations from large language models. In *CVPR*, 2023.

[157] Yi Zhou, Connelly Barnes, Lu Jingwan, Yang Jimei, and Li Hao. On the continuity of rotation representations in neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[158] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models, 2023.

[159] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv*, 2023.

[160] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon A. J. Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. In *SIGGRAPH*, 2004.