Camera Keyframing with Style and Control

HONGDA JIANG, CFCS, Peking University
MARC CHRISTIE, University Rennes, Inria, CNRS, IRISA
XI WANG, University Rennes, Inria, CNRS, IRISA
LIBIN LIU, CFCS, Peking University
BIN WANG*, Beijing Institute for General Artificial Intelligence
BAOQUAN CHEN†, CFCS, Peking University

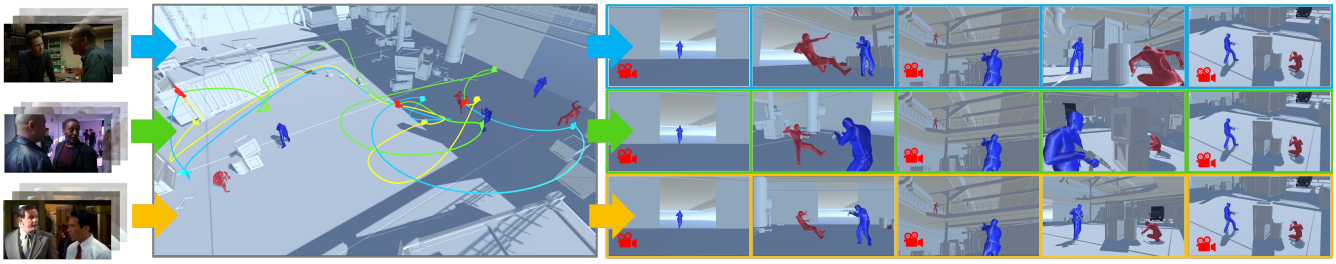

Fig. 1. Our proposed deep-learning framework for camera keyframing offers both high-level style specification and low-level keyframe control. The designer specifies a desired camera style extracted from a reference movie clip (left side) together with a set of camera keyframes as constraints (red cameras in the middle image), and our framework automatically generates in-between camera motions which comply with the specified style while satisfying the keyframe constraints (snapshots on the right). As displayed, different input styles correspond to different camera motions.

We present a novel technique that enables 3D artists to synthesize camera motions in virtual environments following a *camera style*, while enforcing user-designed camera keyframes as constraints along the sequence. To solve this constrained motion in-betweening problem, we design and train a camera motion generator from a collection of temporal cinematic features (camera and actor motions) using a conditioning on target keyframes. We further condition the generator with a *style code* to control how to perform the interpolation between the keyframes. Style codes are generated by training a second network that encodes different camera behaviors in a compact latent space, the *camera style space*. Camera behaviors are defined as temporal correlations between actor features and camera motions and can be extracted from real or synthetic film clips. We further extend the system by incorporating a fine control of camera speed and direction via a hidden state mapping technique. We evaluate our method on two aspects: i) the capacity to synthesize style-aware camera trajectories with user defined keyframes; and ii) the capacity to ensure that in-between motions still comply with the reference camera style while satisfying the keyframe constraints. As a result, our system is the first style-aware keyframe in-betweening technique for camera control that balances style-driven automation with precise and interactive control of keyframes.

CCS Concepts: • **Computing methodologies** → **Procedural animation**.

Additional Key Words and Phrases: Virtual Cinematography, Camera Behaviors, Machine Learning

ACM Reference Format:
Hongda Jiang, Marc Christie, Xi Wang, Libin Liu, Bin Wang, and Baoquan Chen. 2021. Camera Keyframing with Style and Control. *ACM Trans. Graph.* 40, 6, Article 1 (December 2021), 13 pages. https://doi.org/10.1145/3478513.3480533

*corresponding author
†corresponding author

Authors' addresses: Hongda Jiang, jianghd@pku.edu.cn, CFCS, Peking University; Marc Christie, marc.christie@irisa.fr, University Rennes, Inria, CNRS, IRISA; Xi Wang, xi.wang@inria.fr, University Rennes, Inria, CNRS, IRISA; Libin Liu, libin.liu@pku.edu.cn, CFCS, Peking University; Bin Wang, binwangbuaa@gmail.com, Beijing Institute for General Artificial Intelligence; Baoquan Chen, baoquan@pku.edu.cn, CFCS, Peking University.

0730-0301/2021/12-ART1 $15.00
https://doi.org/10.1145/3478513.3480533

1 INTRODUCTION

Designing a camera trajectory in a virtual environment is a demanding task generally achieved by a skilled artist through a sequence of keyframe refinements (placing camera keyframes and tangents, checking results visually, and moving the keyframes until completion). In the literature, only a few techniques have been proposed to ease this design stage. Based on algebraic methods [Blinn 1988], visual servoing [Marchand and Courty 2000], motion planning [Oskam et al. 2009] or numerical optimization [Huang et al. 2016; Olivier et al. 1999], these techniques generally provide a good level of automation at the cost of losing fine-grain user control. Despite these advances, keyframing remains one of the main methods used to produce animations as pointed out in Zhang and van de Panne [2018]. Traditional keyframe interpolators however remain agnostic to the type of motion, stylistic influence, or what they may be linked to, which makes keyframing a labor-intensive and challenging process. An underlying key issue in the design of interactive or automated

camera control techniques also stems from the difficulty to encode many cinematographic principles and rules which guide the design of camera motions and placements [Galvane et al. 2015b].

Cinematography is indeed by nature an empirical process which rules and conventions have been forged through years, tailored by experience and creativity. Rules also evolve over time and principles are often deconstructed, altered or violated to convey specific narrative intentions. As a result, the selection of an appropriate combination of rules and conventions to place and move a camera requires to account for a broad set of factors ranging from low-level geometric features (visibility, lighting, composition) to high-level cognitive and emotional dimensions, all which increase the challenging nature of such a task.

In opposition to approaches that would try to encode rules [Galvane et al. 2015b], a few data-driven techniques have displayed qualitative results in a specific class of problem such as drone cinematography [Bonatti et al. 2020b; Huang et al. 2019a,b] where relations between character poses (or actions) and camera motions are learned from drone film footage. To address ambiguities (different camera trajectories for similar character motions or actions), approaches have either tagged motions by hand according to styles of motion [Huang et al. 2019b], used reinforcement techniques [Bonatti et al. 2020b] or one-shot learning [Huang et al. 2019a]. In the general case of virtual cinematography, a possibility only recently explored [Jiang et al. 2020] is to mimic camera *behaviors* extracted from film clips. A camera behavior is defined as a temporal correlation between camera trajectories and character/actor features such as distance, relative orientation and size. Jiang *et al.* [2020] propose to learn these correlations and handle ambiguities with a Mixture-of-Experts (MoE) approach [Jacobs et al. 1991] where different networks (the experts) train on different sections of the dataset to avoid network collapsing. Interestingly, such approaches address the problem of implicitly encoding elements of *cinematographic style* within a latent space representation, and throughout this paper, we will refer to a *cinematographic style* as the latent encoding of a camera behavior.

Yet, despite some degree of control offered through style specification or a selection of reference clips to be reproduced, such data-driven approaches do not account for constraints on the trajectories such as handling collisions, or ways to force camera positions at specific locations and angles. Indeed, while such learning-based approaches excel in generalizing from a wide range of examples – and in embedding intrinsic knowledge in compact latent spaces – the degree of user control they offer remains limited. This central question of *controllability* in learning has however been addressed in approaches such as motion in-betweening for character animation [Harvey et al. 2020; Zhang and van de Panne 2018] where user-designed keyframes guide the interpolation, giving the designers an additional degree of control and greater precision. A number of data-driven motion prediction approaches [Holden et al. 2017] can also integrate constraints interweaved within the learning stage. Current approaches however do not account simultaneously for multiple styles (in motion) coupled with constraints. Furthermore, this general question of controllability in learning has not been addressed in camera control, and while many automated camera control techniques exist, none offer fine-grain control with high-level style specification.

In this paper, we propose a framework for camera control authoring which enables both an interactive level of control through the specification of keyframe constraints, and an automated level of control through the specification of high-level camera styles. In that, we follow the general recommendation when introducing automation in creative tasks: balancing *computer assistance* with *feeling in control* [Hösl 2019]. The proposed framework is designed to synthesize animations from user-specified dense or sparse keyframes as in Zhang and van de Panne [2018] and yet offers the control and exploration of different camera motions along the animation as in Jiang *et al.* [2020].

The design of such a framework requires addressing a central challenge which is how to handle the simultaneous satisfaction of low-level geometric keyframe constraints and high-level camera style specifications which often are in contradiction. To address this challenge, we draw inspiration from the camera style latent space representation of Jiang *et al.* [2020]. Rather than following a Mixture-of-Experts approach (MoE) where separate experts are trained and balanced through a gating network, we rely on the design of, first, a Long Short-Term Memory (LSTM) gating network (the extractor) trained to extract and identify camera behaviors (geometric correlations of motions) from film sequences. These behaviors are encoding in a latent space representing camera styles. We then design a second LSTM network (the generator) in an autoregressive way whose role is to generate a camera trajectory conditioned by (i) a given style in the latent space and (ii) user-designed keyframe constraints. The training is performed by sampling keyframes on both synthetic and real datasets of cinematic features and using a time-to-arrival embedding to ensure keyframe satisfaction. The sampling is designed to ensure that dense and sparse keyframe constraints can be satisfied. Finally, a dedicated mapping network is trained to warm start the hidden states of the generator LSTM, enabling an additional control of camera velocities by the designer and improving the training loss.

The contributions of this paper are therefore:

- the provision of an interactive camera keyframing tool which mixes low-level control through user-specified keyframes and high-level control of in-between motions driven by camera behaviors extracted from example film clips;
- the design of a two-fold learning framework (gating + generation) which improves over previous work by exploiting a simpler structure than the Mixture-of-Experts and is trained on a dataset of motions extracted from the MovieNet repository [Huang et al. 2020];
- a validation of the system through the design of specific metrics to ensure that a proper balance between style control and keyframe control is achieved.

2 RELATED WORK

2.1 Keyframe-based animation control

As highlighted in Zhang and van de Panne [2018], given the importance of keyframing in general animation tasks and the need to assist designers in automatically or interactively filling in gaps

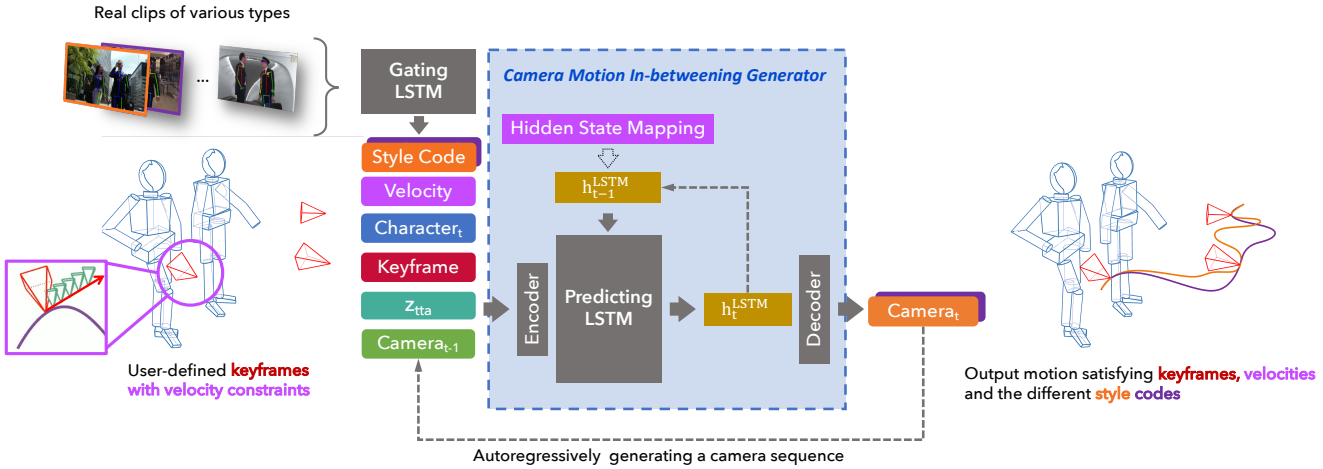

Fig. 2. Our proposed framework for learning camera together with keyframe constraints composed of a camera behavior extractor (Gating LSTM), which extracts camera behaviors from reference clips, and a camera motion generator, which generates camera trajectories that both meet the camera behaviors and required keyframe constraints, speed and directions.

of animation, a number of techniques have been proposed to constrain the motions such as using space-time constraints [Witkin and Kass 1988], physical models for realism [Faloutsos et al. 2001], or probabilistic approaches [Min and Chai 2012].

The problem is actually similar to *curve-fitting* with a sparse dataset and is often addressed through optimization (see the tangent-space technique [Ciccone et al. 2019]). In the specific case of camera keyframing, optimization has been used for tasks such as designing overviews of car motions [Huang et al. 2016], generating in-between animations that ensure visual properties [Lino and Christie 2015] or generating camera rails between specified viewpoints [Galvane et al. 2015a]. In Lino and Christie [2015], they rely on the Toric Space representation to blend visual properties (camera composition, camera angle and distance to characters) and maintain the blended properties over time. Yet, the system does not handle style characteristics, nor can it handle very sparse keyframes.

In character animation, by relying on the availability of rich datasets, approaches such as *motion matching* perform a search in motions to match constraints such as keyframes or trajectories [Büttner and Clavet 2015]. This problem of motion in-betweening has also been addressed through learning. Zhang *et al.* [2018] designed a Recurrent Neural Network (RNN) that is conditioned on target keyframes. By learning motion characteristics, together with a dedicated blending function to ensure keyframe matching, the approach provides both precise keyframe control and respects motion characteristics from the database, yet does not offer control in the style of interpolation. More recently, Harvey *et al.* [2020] extended the state of the art motion predictors which using RNNs, by adding a time-to-arrival embedding to inform the network of the progression towards a keyframe constraint, and a scheduled target noise vector to enforce stochasticity in the transitions. In addition, a Generative Adversarial Network (GAN) is exploited to further improve the quality of the transitions. This method is limited to gaps of short motion intervals, say 40 frames, due to cyclic motions and despite being able to generate some variety in motions between the gaps (by using a noise vector), the degree of control on the style remains limited.

In contrast, we first design an LSTM gating network which identifies variations in behaviors of the camera, and then design an autoregressive prediction network that is conditioned on the keyframes and the style label. As a result, not only can keyframes be enforced, but designers can constrain the camera behaviors between the keyframes, by either explicit specification, or extraction from a reference clip. Given this ability to comply to behaviors, motion sequences can be generated with far less keyframes than traditional approaches.

2.2 Camera behavior mimicking and learning

As demonstrated through a number of contributions [Galvane et al. 2015b], the automated or interactive design of camera motions requires manual encoding of cinematographic rules and conventions which are numerous, and in some cases contradictory. Among the combinatorial possibilities in placing, moving and cutting between cameras, only few subsets are valid, and these valid subsets characterize distinct cinematic styles. Galvane *et al.* [2015b] relied on the manual design of weights between rules, independently of events in the scene. Other optimization-driven frameworks encoded camera motion characteristics (smoothness, optic flow) for specific requirements such as route overviews [Huang et al. 2016] or virtual walkthroughs [Argelaguet and Andujar 2010], with also the burden of balancing weights of different rules through experiments.

In such a context, the idea of automatically learning how cameras should be placed, moved and edited from real footage is appealing, but requires to address challenges such as (i) extracting cinematic features, (ii) accounting for stylistic variations, and (iii) adapting the learned knowledge to new environments. By addressing a specific context - drone cinematography - Huang *et al.* [2019b] have been the first to propose the automated learning of camera motions.

After a careful selection and manual classification in four styles of real drone footage following some characters, the authors proposed an LSTM structure to learn the correlation between the characters motion and the drone motion. Extensions were proposed to account for scene background information in the correlation [Huang et al. 2019c] or use one-shot learning techniques to avoid manual style classification [Huang et al. 2019a]. In the same context, the learning of artistic principles for drone cinematography was addressed by deep Reinforcement Learning (RL) [Gschwindt et al. 2019]. Viewpoint selection was supervised by a deep RL agent trained and evaluated on synthetic content. Later contributions also relied on crowd-sourced synthetic contents [Bonatti et al. 2020a] to learn by regression a *synthetic descriptor space*, establishing correlations between low-level shot parameters (drone angle, distance, velocity) and perceptually meaningful parameters such as *exciting, enjoyable, establishing, revealing*.

By addressing virtual cinematography applications, Jiang *et al.* [2020] proposed the notion of camera behaviors as the temporal evolution of camera features in correlation with scene features (actors on-screen positions, distances between actors, relative orientations), and showed how such behaviors could be learned by automatically extracting features from film footage. To address ambiguities in the dataset (different camera motions for similar scene features), the authors propose a Mixture-of-Experts (MoE) deep learning approach, which yields a latent camera style space that can be used to both identify camera behaviors from real footage, and drive camera motion predictors. Yet, the degree of control for designers is limited to choosing reference clips from which style is extracted. In contrast, our approach provides designers with additional flexibility through the specification of keyframe constraints, while enforcing selected camera styles between keyframes.

3 OVERVIEW

Our objective is to provide designers with a camera control tool which can encode camera styles from example clips and transfer them to a given 3D animation while accounting for to user-designed keyframe constraints. In a nutshell, we want to solve a constrained style-aware camera motion in-betweening problem.

3.1 Camera behaviors

A camera behavior depicts a specific correlation between character and camera motions. In this paper, we mathematically define this correlation using the *cinematic feature space* proposed by Jiang et al. [2020]. The *cinematic feature space* for a given frame is composed of character features $\mathbf{x}^v$ and camera features $\mathbf{x}^c$, defined as:

$$\mathbf{x}^v = \{d_{AB}, s_A, s_B, s_{AB}, M\} \in \mathbb{R}^5, \quad (1)$$

$$\mathbf{x}^c = \{\mathbf{p}_A, \mathbf{p}_B, \theta, \phi\} \in \mathbb{R}^6, \quad (2)$$

where A and B refer to left and right characters on the screen respectively, d_{AB} is the 3D distance between the two characters, s_A (resp. s_B) represents the angle between line AB and the front vector orthogonal to the segment between character A (resp. B) shoulders defined as the relative orientation of the torso to line AB. s_{AB} describes the angle between the characters shoulders orientations. M is a binary variable indicating the main character of the sequence (usually the character occupying a larger area in the shot). The camera pose is expressed in the Toric space coordinate [Lino and Christie 2015] which is an expressive and compact local representation on two given targets. $\mathbf{p}_A, \mathbf{p}_B$ represent the normalized on-screen position of two characters, θ and ϕ are two parametric angles representing the yaw and pitch angles towards the targets in 3D space (see Fig. 3 for details). Based on the preceding definitions, a keyframe constraint can be represented as a camera feature $\mathbf{x}^c$ at a given time; while a camera behavior is the evolution of cinematic features in a period of time. We denote $\mathbf{X}^v$ a sequence of character features and $\mathbf{X}^c$ a sequence of camera features.

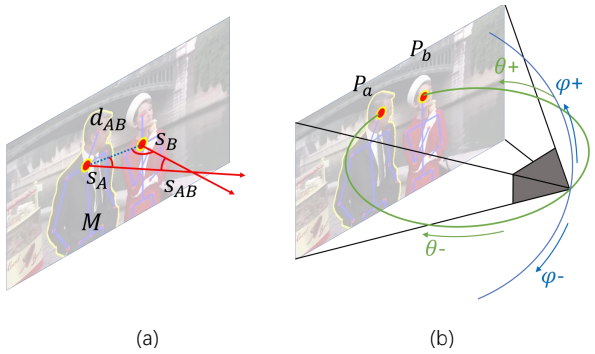

Fig. 3. Illustration of cinematic features: (a) the character features, such as inter-character distance, absolute and relative orientations on yaw direction of shoulders and (c) the camera pose expressed in Toric space coordinates that describes a relative pose using framing features (the 2D on-screen position of characters) together with pitch and yaw angles (θ, φ).

3.2 Pipeline

We design a two-stage pipeline (see Fig. 2) comprising a *camera behavior extractor* which extracts a camera behavior from a real film clip and encodes it as a style code in a latent space representation and a *camera motion generator* which is a predicting generative LSTM module conditioned on the given style code as well as user-designed keyframe constraints.

The input of our trained system is a 3D animation with two characters, a number of user-defined camera keyframes along the animation, and a reference video sequence to specify the behavior. We first estimate the sequence of cinematic features (character features and camera motion) from the reference video as in Jiang *et al.* [2020] and input the extracted information to the *camera behavior extractor* to identify a corresponding style code. Then, we deliver the style code to the *camera motion generator* together with the immediate next keyframe constraint, the last camera pose, as well as the target 3D character animation. The system outputs the current camera pose and feeds it back as the input of *camera motion generator* to predict the camera trajectory in an autoregressive manner. The camera poses generated at different stages of the system are all expressed in Toric space coordinates which can then be applied to a 3D animation to yield the final 6D camera pose (position and orientation).

In the following, we introduce our main method and the training details in Section 4. The ablation study and results are demonstrated

in Section 5 and Section 6, before the limitations and discussion in Section 7.

4 CAMERA MOTION IN-BETWEENING

We first design a camera behavior gating network able to encode different geometric behaviors from reference clips in a latent space of camera styles. We then design a camera motion generator network to generate trajectories controlled by keyframe constraints and style labels, and illustrate how our networks are trained simultaneously.

4.1 Camera behavior gating network: the extractor

Our gating network acts as a camera style selector through the construction of a low dimensional manifold of camera styles. It is designed as an encoder network using a LSTM backbone structure. The sequence of cinematic features $\mathbf{X}^v, \mathbf{X}^c$ extracted from a film clip is fed to the gating LSTM network with the hidden state initialized to zero state. The last vector of the LSTM output is delivered to a fully connected module to obtain a low dimensional style code $\mathbf{z}^c \in \mathbb{R}^4$ (see appendix A.1 for more details on the network architecture).

We do not add any constraints to $\mathbf{z}^c$ since the normalization operations such as softmax or KL-divergence tend to limit the capacity of style codes to represent different behaviors, and can require additional weight fine-tuning (*e.g.* KL-divergence loss). Despite its low dimension, the latent space provides a clear separation of behaviors. This is illustrated in Fig. 4 by displaying the PCA representation of the style code where colors represent the four behaviors (*direct, relative, side and orbit*) found in our synthetic dataset. The symmetric distribution reflects left or right main character on the screen.

4.2 Camera motion predicting network: the generator

Given a style label from the latent space, our proposed camera motion generator is in charge of predicting the current camera pose based on the current character feature vector, the next keyframe constraint and the past camera pose. We use LSTM as the backbone structure as well, and the system can be formulated as:

$$\mathbf{x}_i^c = f(\mathbf{z}^c, \mathbf{x}_{i-1}^c, \mathbf{x}_i^v, K, z_{tta}, h_{i-1}^{\text{LSTM}}), \tag{3}$$

where $\mathbf{x}_i^c$ and $\mathbf{x}_i^v$ represent camera and character feature at frame i respectively; $\mathbf{z}^c$ depicts the style code of the reference camera behavior; $K \in \mathbb{R}^{11}$ represents camera and character feature at the next keyframe; time-to-arrival embedding $z_{tta} \in \mathbb{R}^{128}$ indicates the number of frames to the next keyframe and h_{i-1}^{LSTM} is the hidden state of LSTM network at frame $i-1$. All the above features except the hidden state are concatenated and entered into an encoder which is composed of two fully connected layers with ReLU activation in the succession of each. The LSTM network is fed with the output of the encoder, and the hidden state h_{i-1}^{LSTM} from the previous iteration. The network outputs a hidden state vector $h_i^{\text{LSTM}} \in \mathbb{R}^{256}$ to be reused in the next iteration. The decoder is composed of two fully connected layers with ReLU only after the first one. The predicted camera poses are generated autoregressively: each predicted frame serves as input for the next LSTM iteration.

We also designed our system to handle both dense and sparse keyframes. Inspired by Harvey *et al.* [2020], we use the time-to-arrival signal z_{tta} to measure the time between the current frame and the next target keyframe. This signal gives the network a hint on the importance of current keyframe constraint. The time-to-arrival signal embedding is defined as a 2-dimension vector z_{tta}:

$$z_{tta,2i} = sin(\frac{tta}{b^{2i/d}}), \tag{4}$$

$$z_{tta,2i+1} = cos(\frac{tta}{b^{2i/d}}), \tag{5}$$

where tta is the number of time steps to the next keyframe and d is the dimension of the input embedding. The basis value b influences the rate of change in frequencies along the embedding dimension.

The z_{tta} embedding provides a smooth and continuous positional encoding of the current frame. However, to reduce the influence of z_{tta} with distant keyframes, tta is capped to a threshold value tta_{max}. We empirically set tta_{max} to 200. This enables our system to handle both sparse and dense keyframes (see companion video), and favors the enforcement of the camera style between distant keyframes.

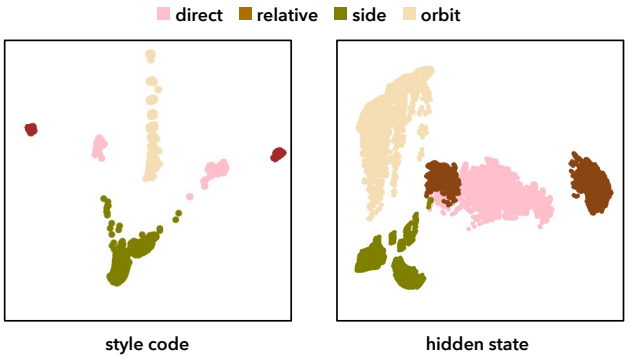

Fig. 4. To validate the proper identification of camera behaviors, we illustrate PCA results of the style code and hidden state vector distribution. Both distributions display a good separation among camera behaviors, tagged for the display only and not used in the training. Behaviors are coloured according to the tags in the synthetic dataset: direct, relative, side and orbit.

4.3 Velocity control using hidden state mapping

For some particular types of camera behaviors, camera features $\mathbf{x}^v$ cannot be fully determined by character features $\mathbf{x}^c$. Extra parameters are required, typically to encode motions with phase and frequency information for some cyclic orbit tracks in which the camera oscillates from one side of a character to the opposite side of the other character. However, utilizing the combination of latent style code $\mathbf{z}^c \in \mathbb{R}^4$ and two keyframes as input is not sufficient to enforce the prediction LSTM network to generate the desired camera motion, especially when the hidden state of a LSTM network is initialized to zero or a random sample from a Gaussian distribution.

To improve our prediction network and extend the range of camera behaviors it recognizes, we extend our system by adding a hidden state mapping module $H(\cdot)$ which maps a sequence of five camera configurations $\mathbf{x}_0^c..\mathbf{x}_4^c$ (representing the desired local velocity) and a style code z^c to a specific starting hidden state, defined by:

$$h_0^{\text{LSTM}} = H(z^c, \langle \mathbf{x}_0^c, ..., \mathbf{x}_4^c \rangle). \tag{6}$$

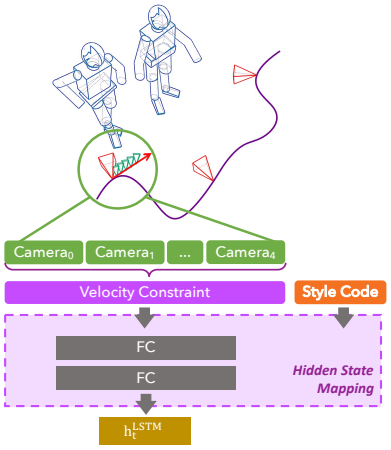

Fig. 5. By learning the mapping from a style code and several camera frames (5 in this paper) towards the starting hidden state of our autoregressive generator, we provide an additional degree of control for the users through the specification of camera velocities.

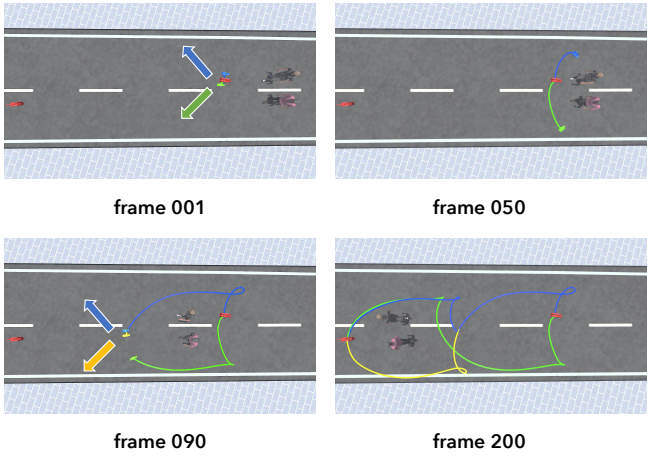

Fig. 6. The proposed system enables designers to specify keyframes with initial velocities. In this example, the same keyframe positions and camera style code is used. As displayed, the resulting trajectories are guided by the different velocity directions defined at the starting keyframe (time 001) and the mid keyframe (time 090) respectively. This is achieved by updating the LSTM hidden state through a dedicated network which maps velocities with hidden states.

As demonstrated in Fig. 5, the mapping function $H(\cdot)$ has two fully connected layers with ReLU activation after the first one.

The hidden state of a LSTM network is normally initialized to zero or a random sample from a Gaussian distribution. However, the hidden state of our generator is initialized or reset through a learnable hidden state mapping function $H(\cdot)$. This design helps to resolve the ambiguity issue on camera behavior classification and endows our system with the ability to control camera direction and speed at any location along the trajectory by resetting the LSTM hidden state. More specifically, by simply breaking the original hidden state feedback loop in LSTM generator network and applying a 5-frames long desired camera motion to $H(\cdot)$, artists can control camera direction and speed at any location along the trajectory. A concrete example is displayed in Fig. 6 where different velocities at initial frame (001) and middle frame (090) are applied. It is notable that the hidden state mapping is unaware of the history information.

4.4 Training and loss

We trained all three modules simultaneously in an end-to-end fashion with an objective formulated as a weighted sum of two losses, defined as:

$$L = L_{rec} + L_K = \frac{1}{n}\sum_{i=1}^{n} ||\hat{\mathbf{x}}_i^c - \mathbf{x}_i^c||_2 + \frac{1}{||\mathbf{K}||}\sum_{k\in\mathbf{K}} ||\hat{\mathbf{x}}_k^c - \mathbf{x}_k^c||_2, \quad (7)$$

in which the **reconstruction loss** L_{rec} measures the difference between predicted camera features $\hat{\mathbf{x}}^c$ and ground truth camera features $\mathbf{x}^c$; while the **keyframe loss** evaluates the difference of camera features between the generated keyframes and the ground truth; and $\mathbf{K}$ is the set of keyframes.

We use the Adam [Kingma and Ba 2015] adaptive gradient descent algorithm. Training is performed for 200 epochs and takes around 4 hours on an NVIDIA Tesla V100S GPU with a batch size of 1024. We use the exponential learning rate policy with the base learning rate set to 0.001 and decay 0.97 after each epoch.

4.5 Dataset

Our work relies on the creation of a hybrid dataset composed of 2,640 synthetic sequences and 17,700 movie sequences. The synthetic data is the same one as Jiang *et al.* [2020] generated from 30 manually designed 3D animated scenes with a length of 1,500 frames each. Four well-known cinematography behaviors (direct track, side track, relative track and orbit track) are implemented in the dataset.

The movie data is extracted from the MovieNet dataset [Huang et al. 2020] which consists of 1,100 movies and 1,600,000 clips. We estimated the cinematic features from a subset of movie clips using the *cinematic feature estimator* [Jiang et al. 2020] (a convolutional neural network which regresses the cinematic features from 2D skeleton motions). For the movie dataset, we first filter the sequences according to their number of characters and clip length. A visibility requirement is also applied to ensure the target characters are constantly present on the screen. In order to avoid erroneous estimation and imprecise behavior recognition caused by abnormal data samples, we exclude the clips with unusual character sizes and extreme recording lengths (overly short or long). We finally select 17,700 movie clips for a total of 4,900,000 frames. A smoothing filter (average sliding window) was applied to the output of the *cinematic feature estimator* to reduce the noise in the data. Extractor, generator and mapping modules are all trained with this same dataset, yet dedicated treatments are applied for some modules.

Preparing sequences for the extractor. The identification of camera behaviors requires a long enough sequence of cinematic features. As reported in Jiang *et al.* [2020], 300 frames is sufficient and we therefore randomly sample sub-sequences of greater length. To accelerate both training and testing procedures, we reduce the number of input frames using a downsampling technique (see Section 4.4). Considering the continuous nature of a camera motion,

this downsampled frame rate has a limited influence on learning performance.

Preparing sequences for the generator. For the generator, we extract segments of cinematic features at random locations, with a number of frames larger than $tta_{max} = 200$ frames. Then two camera frames are arbitrarily selected from each segment and will represent the starting and ending keyframes from which to learn. As a result, training data covers different lengths between keyframes as well as different locations of keyframes along the trajectories.

While existing motion in-betweening techniques [Harvey et al. 2020; Zhang and van de Panne 2018] input samples at every frame, we propose in practice to downsample the input signal (both camera and character features) by only considering every five frames. Resulting camera motions are smoothed with low frequencies, and filled with cubic interpolation when required. Since the autoregression is computationally intensive, our downsampling strategy can significantly accelerate the camera prediction process with a limited influence on results.

5 ABLATION STUDY

5.1 Evaluating the hidden state mapping

In order to justify the effectiveness of our hidden state mapping function $H(\cdot)$, we compare our method with three other hidden state initialization methods. We first test the zero and Gaussian-initialized hidden states which are commonly used without extra knowledge of the starting camera velocity. We then design another framework that warm starts the hidden state from zero by inputting the first five frames of ground truth camera and character features, instead of using the predicted features, both during the training and testing stage.

We compare the four methods by reporting trajectory error on the synthetic evaluation dataset. As shown in Tab. 1, our method outperforms the others for all behaviors; and the warm start scheme exhibits noticeable performance gain on orbit track compared with zero and Gaussian initialization. These results confirm the efficiency of the hidden state mapping module in reducing ambiguities in camera behaviors.

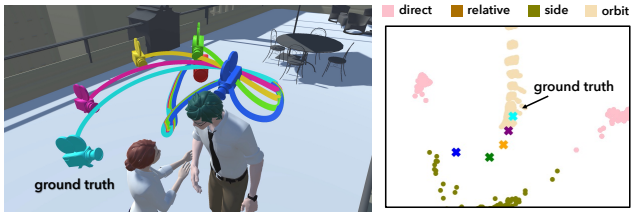

Fig. 7. User-specified keyframes are placed at increasingly larger distances from the trajectory of a given style. As displayed, our system adapts well to the keyframes. We re-extract the style codes from the generated trajectories (shown with crosses in the PCA representation on the right part of the figure). As displayed our system moves from the given style to adapt to the keyframe constraints.

5.2 Evaluating the satisfaction of style and keyframes

In the following, we test the capacity of our method to address conflicting situations that arise when specified keyframes are distant from what is expected in a given camera style.

A qualitative evaluation is displayed in Fig. 7 in which we use 5 colors to represent 5 different situations with identical starting keyframe but different ending keyframes (represented as the colored camera icons). Among them, the keyframe and curve with cyan color represent the ground truth (*i.e.* the unconstrained trajectory generated by the specified style). The other 4 curves are generated using the same style code with specific ending keyframes. As displayed in Fig. 7, all the trajectories pass through their corresponding ending keyframes. We then project back the generated trajectories in the latent camera style space (using the gating network). The corresponding style codes gradually drifts away from the specified style code. Both the resulting trajectory and latent space representations display a smooth and consistent change.

We further conduct a quantitative comparison with Jiang et al. [2020] on the synthetic dataset.As shown in Fig. 8, our evaluation includes two steps:

i) we extract the style code z^c from ground truth trajectories x^c with the gating network, and use the style code to generate the trajectories $x^{c\prime}$ with the same animation but *displaced* keyframes. The *displaced* keyframes are generated by randomly perturbing keyframes on ground truth trajectories with different magnitudes of noise and clamping them to a valid camera pose.

ii) we then extract the style code z_r^c from $x^{c\prime}$ and generate the trajectories x_r^c with the same animation and the ground truth keyframes.

We designed two metrics, latent normalized silhouette distance (SD) and trajectory distance (TD) to respectively evaluate how style (SD) and behaviors (TD) are preserved. The definition of SD is as:

$$SD(Z_c^r, Z_c) = \frac{S(Z_c^r, Z_c)}{S(Z_c, Z_c)}, \tag{8}$$

where the distance function $S(U, V)$ describes the average Euclidean distance of vectors between set U and set V; Z_c and Z_c^r are collections of style codes z_c and z_c^r respectively. All the z_c in Z_c should belongs to same style category, and the $SD(Z_c^r, Z_c)$ indicates the impact of keyframe perturbation on style.

Another metric $TD = ||\mathbf{x}^c - \mathbf{x}_r^c||$ tends to measure behavior similarity through trajectory discrepancy. For a valid evaluation, $\mathbf{x}_r^c$ needs to be generated using identical keyframe constraints as $\mathbf{x}_c$, and with the latent code compatible with the augmented keyframe constraints. To achieve compatible latent code, we correct $x^{c\prime}$ firstly using a linear interpolation as:

$$\mathbf{x}_i^{c\prime} += (1 - \lambda_i)\Delta\mathbf{x}_{i-1}^{c_k} + \lambda_i\Delta\mathbf{x}_i^{c_k}, \tag{9}$$

where $\Delta\mathbf{x}_{i-1}^{c_k}$ and $\Delta\mathbf{x}_i^{c_k}$ represent keyframe distance of the immediate left and right keyframes of frame i, and $\lambda_i \in [0, 1]$ represents the weight. Then, the compatible z_r^c is extracted from the corrected trajectory $x_c{}'$.

We compare three methods in Tab. 2. The first one is our proposed method with the trajectories generated following the preceding instructions; the second one comes from Jiang *et al.* [2020] with behavior constraint only; in order to make a fair comparison with Jiang *et al.* [2020], the third one labelled as 'only style' is a modified

	Direct track	Relative track	Side track	Orbit track	Average
Our method	**0.107**	**0.050**	**0.025**	**0.049**	**0.057**
Zero hidden state	0.162	0.081	0.045	0.263	0.138
Gaussian hidden state	0.146	0.074	0.045	0.263	0.132
Warm start hidden state	0.173	0.088	0.071	0.116	0.112

Table 1. Influence of different hidden state initializations on trajectory reconstruction error. The error is expressed as the difference in Toric space coordinates between the generated trajectory and the ground truth trajectory.

version of our proposed method. $x^{c\prime}$ in step i) is generated using ground truth keyframes and then refined to meet the augmented keyframes through linear interpolation. The result in Tab. 2 shows that our proposed method could maintain the behavior when the keyframe difference is not too far from the ground truth. With quite different keyframes, our method better satisfies the behavior specification compared to Jiang *et al.* [2020].

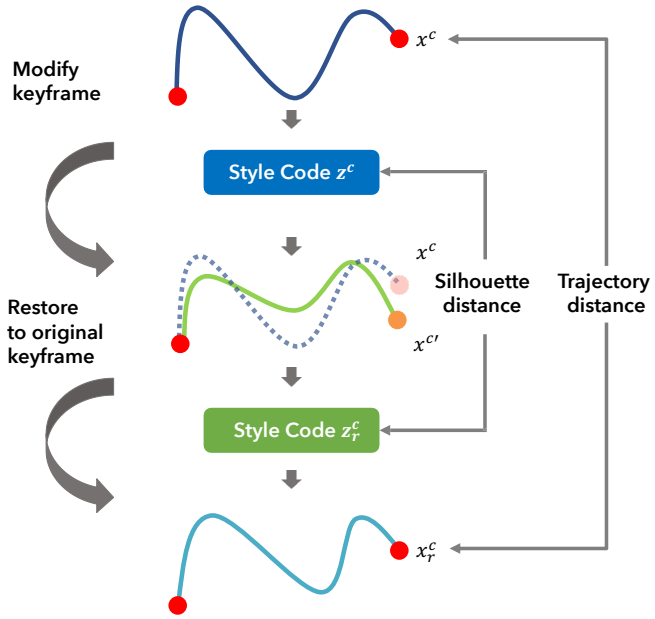

Fig. 8. Metric for evaluating our behaviors preservation capacity when conflicting with distant keyframes. Two metrics are introduced: i) Silhouette distance (SD) is style code distance before and after keyframe modification; ii) Trajectory distance (TD) means the difference in trajectories between the original one and the one which reuses the style code from the modified keyframe.

5.3 Evaluating the influence of movie data

During the training process, the sampled keyframes are all taken from the ground truth trajectories. However, the risk of over-fitting increases if we only train with synthetic data. As real movie data contains more complex and natural camera behaviors, we exploit them to increase the variety in camera behaviors when training, and hereby demonstrate their influence.

We compare the testing error in evaluation dataset and also evaluate SD and TD metrics with the process illustrated in Fig. 8. In contrast with Section 6.2, we do not apply trajectory correction via linear interpolation for a strict keyframe constraint, but calculate the keyframe distance (KD) as another metric to measure the capacity to meet the keyframe. Tab. 3 demonstrates that the movie data helps to increase the robustness and the capacity of the network to meet both behavior and keyframe constraints.

6 RESULTS AND EXPERIMENTS

6.1 User interface

To apply our work in practice, we deploy the model in a custom plugin in Unity 2019. With a given 3D animation, the user can set and edit the keyframes on the timeline. The UI then automatically generates the essential information and communicates with the trained camera motion predictor by ZeroMQ to produce trajectories from different style codes and given keyframes.

As shown in Fig. 9, we provide trajectories with camera behaviors from user specified movie clips or default synthetic trajectories as reference clips. The user can select trajectories with different behaviors between keyframes (D). Furthermore, the user can adjust the results by adding or removing keyframes (C). The running speed of our system is fast enough for animation design. When working with a new scene, we need to process the scene information in 200 frames per character per second. After that, we can generate camera trajectories with user satisfied keyframe constraints in more than 2,000 frames per second.

Keyframe editing and trajectory generation. In the user interface, the user can set keyframes directly on the timeline (B). To adjust the camera view of a selected keyframe, user can assign the target characters on which the keyframe focus and set the camera pose. After choosing the favored keyframes, the 'generate' button triggers the computation of trajectories. If no specific style code is selected, trajectories with different style codes are generated automatically and listed in the interface for user to select and preview.

Character transition and cutting. For complex scenes with multiple characters, we can specify the two target characters for each individual keyframe which (C in Fig. 9) so to perform transitions between characters. Changing character information is sent to the model and the resulting Toric camera parameters will be applied to the new character targets.

Collision avoidance. Collision avoidance is always a critical issue when artists design animations since the desired camera motion may unexpectedly hit other objects in the virtual scene. Compared

	Metric	0 (radian)	0.1	0.3	0.5	0.9	1.2	1.4	1.6	1.8	2.0
Our method	SD	**1.009**	**1.009**	**1.015**	**1.025**	**1.058**	**1.082**	**1.1**	**1.12**	**1.157**	**1.211**
Our method (only style)	SD	1.013	1.013	1.023	1.043	1.114	1.224	1.338	1.426	1.487	1.534
[Jiang et al. 2020]	SD	1.016	1.012	1.015	1.034	1.099	1.212	1.316	1.401	1.461	1.510
Our method	TD	**0.065**	**0.069**	**0.089**	**0.115**	**0.175**	**0.219**	**0.24**	**0.263**	**0.302**	**0.359**
Our method (only style)	TD	0.065	0.072	0.103	0.139	0.254	0.323	0.379	0.41	0.452	0.475
[Jiang et al. 2020]	TD	0.147	0.239	0.444	0.554	0.804	0.935	0.979	1.026	1.065	1.102

Table 2. Trajectory reprojection loss for different keyframe bias. Latent normalized silhouette distance (SD), trajectory distance (TD). The top row displays the distance from the initial ground truth value θ to the new θ_k value of the given keyframe in radians. By increasing the distance, we measure the capacity of the network to meet the keyframe constraint (metric TD) and the style constraint (metric SD).

	Direct track	Relative track	Side track	Orbit track	Average
Our method	**0.107**	**0.050**	**0.025**	**0.049**	**0.057**
Our method without movie data	0.126	0.056	0.033	0.060	0.069

	Metric	0 (radian)	0.1	0.3	0.5	0.9	1.2	1.4	1.6	1.8	2.0
Our method	SD	**1.009**	**1.009**	**1.012**	**1.019**	**1.040**	**1.060**	**1.075**	**1.095**	**1.126**	**1.168**
Our method without movie data	SD	1.022	1.022	1.024	1.029	1.045	1.070	1.103	1.147	1.195	1.242
Our method	TD	**0.064**	**0.068**	**0.088**	**0.114**	**0.168**	**0.207**	**0.227**	**0.242**	**0.269**	**0.311**
Our method without movie data	TD	0.076	0.078	0.093	0.116	0.177	0.231	0.271	0.321	0.355	0.395
Our method	KD	**0.016**	**0.021**	**0.051**	**0.085**	**0.154**	**0.192**	**0.216**	**0.235**	**0.251**	**0.277**
Our method without movie data	KD	0.026	0.037	0.080	0.125	0.212	0.258	0.281	0.299	0.321	0.351

Table 3. Influence of adding movie data in the training. The above table shows the generated trajectories distance as testing loss in testing dataset, and the bottom table shows the capacity to preserve behaviors with different keyframe and the capacity to meet the keyframe constraints. By increasing the distance between the groundtruth value and a new keyframe, we measure the capacity of the network to meet the keyframe constraint (metric TD) and the style constraint (metric SD).

with the example-driven solution [Jiang et al. 2020], which has to redundantly try multiple references clips to find one without collision, here the designer has the ability to avoid collision by easily inserting a keyframe or forcing the velocity to where no obstacles blind the camera's view.

6.2 Results

Dialogue scene. For this result, we choose a fierce argument sequence (900 frames at 30 fps), and generate camera trajectories under different required camera behaviors and keyframe constraints. We test our method both on style variation and keyframe matching. As illustrated in Fig. 10 and Fig. 11, we demonstrate the camera trajectories with different behaviors using the same keyframes and with the same behavior but different keyframes respectively.

For the former experiment, the main idea is to utilise different camera behaviors to generate motions with the same keyframe constraints in order to highlight that our method is able to converge independently on the behaviors. On the other hand, in the later experiment, we show that our method can fine-tune the trajectories in a low-level fashion and retain the behavior style simultaneously.

Zombie scene. For the zombie scene, we choose a close quarter battle sequence (900 frames at 30 fps), and show the camera trajectory evolution followed by editing the keyframes, as illustrated in Fig. 12. The main challenges in this scene are the complex and fast-paced motions of all 5 characters and the wide-spread obstacles which may cause collision with the camera. In this scene the targets

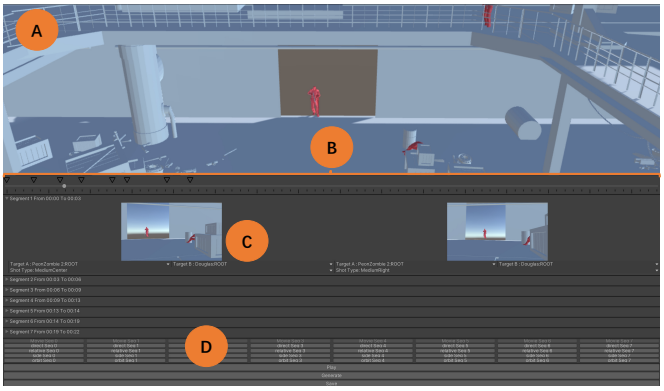

Fig. 9. Our camera trajectory editing interface. The scene view (A) displays the animation. In the timeline (B) the user can add, drag, and delete keyframes (inverted triangles), as well as drag the process of animation. The keyframe editing (C) allows the user to select two target characters, shot view and Toric camera pose at the keyframe. The trajectories selection (D) provides generated camera trajectories with different behaviors for the user to choose. The button on the bottom is used to preview a result (Play), generate trajectories (Generate) and save results (Save).

are different from one keyframe to another to make the camera catch up with the character motions. The sequence was designed by an senior previs artist. By editing the keyframes, he had interactive

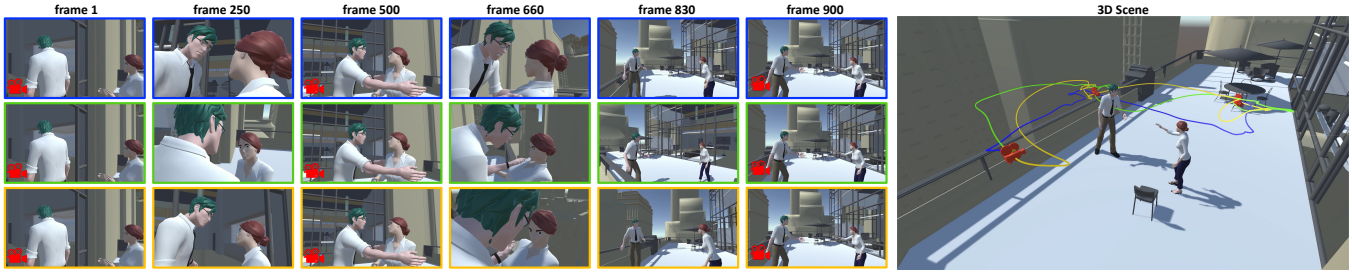

Fig. 10. Experiment with same keyframes and different behaviors: different colors represent different camera behaviors. Frames with red camera icon at the corner refer to keyframe constraints. We observe that constraints are well enforced in the 3D content and in the rendered snapshots.

Fig. 11. Experiment with different keyframes and same behavior: different colors represent different keyframes fed to the system with the same style code. All three sequences belongs to a same style but their trajectories adjust well in response to the required keyframes (frames with different colors of camera icons at their corner).

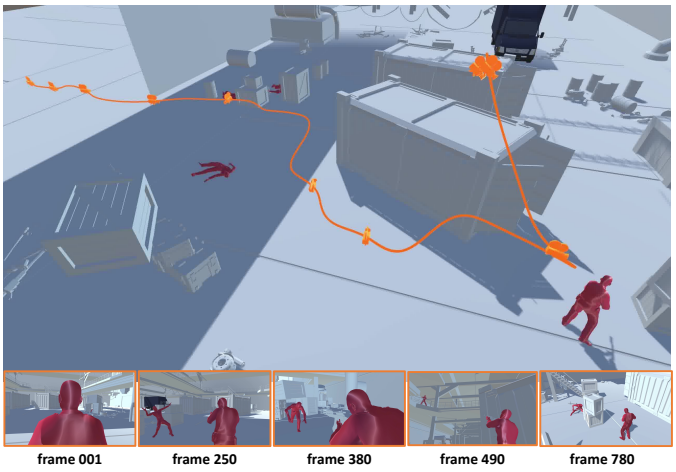

Fig. 12. This figure displays a result designed by an animation artist using only 10 keyframes for a 24 seconds sequence of a zombie fighting scene. We show the keyframes and camera trajectory simultaneously with the rendered animation snapshots.

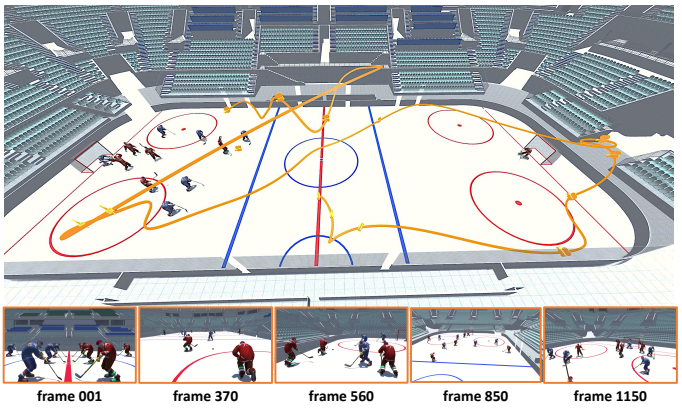

Fig. 13. In this hockey game scenario, our method is able to generate dynamic and qualitative camera motions using only 10 keyframes. Rendered animation snapshots and the overview trajectory are displayed.

control over the trajectories to both avoid collisions with the scene and fine-tune the camera path.

Hockey scene. For the Hockey scene, we choose a hockey game sequence with ten players and two goalkeepers (1300 frames at 30 fps) and show the camera trajectories in Fig. 13. The main challenges in this scene are the fast-paced motion of characters and frequent main characters switch due to the pass of the hockey ball. By editing keyframes, we have the ability to fine-tune the trajectory and increase the pitch angle to deal with the occlusion between players. The final result provides a different experience compared to common way to watch a game.

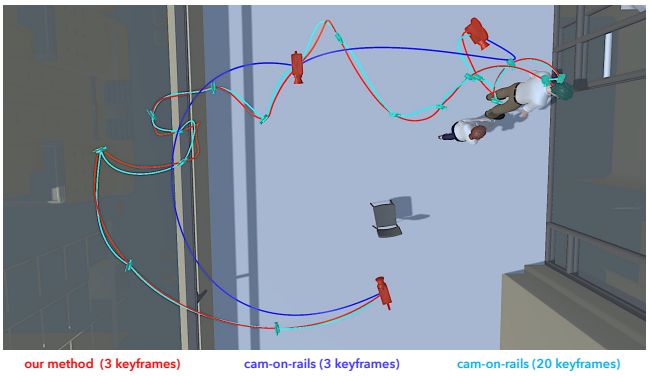

Fig. 14. We show a comparison with the camera-on-rails interpolation method. The figure illustrates that our proposed method (red) can produce dynamic and styled camera motion with only 3 keyframes, whereas the camera-on-rails of 3 keyframes (blue) only interpolates without extra style control between the keyframes. To achieve a similar result w.r.t our proposed method, 20 keyframes are required to match the camera-on-rails method (cyan).

6.3 Comparison with optimization based methods

To demonstrate the efficiency of our approach, we compare with optimization-based methods which are also commonly employed in the offline camera path planing applications. In order to make a fair comparison, we choose the Camera-on-rails [Galvane et al. 2015a] technique as candidate, which handles specifically the framing of two targets also with the aid of a Toric coordinate representation. Given keyframe camera set K, the technique searches for the optimal parameters on a third-degree spline curve (the camera rail) by minimizing a metric expressed on the framing along this trajectory between adjacent keyframes. The metric is expressed as a distance between the spline curve and a curve which perfectly interpolates keyframes' visual properties along the sequence.

The comparison is conducted between three camera motion in-betweening trajectories: i) X^c_{ours} is generated utilizing the proposed method with 3 user-designed keyframes and the desired behavior; ii) X^c_{opt} is generated utilizing the camera-on-rails method with the same keyframes as X^c_{ours}; iii) X^c_{opt} is generated utilizing the camera-on-rails method as well, but with increasing number of keyframes until the generated trajectory reaches a qualitative similarity to X^c_{ours}. In essence, this mimics the process when an artist manually creates a trajectory by setting keyframes and using optimization-based methods to achieve the goal in-betweening trajectories. Results are displayed in Fig. 14 and in our companion video. We observe that the optimization method requires up to 20 keyframes to obtain a relatively similar result. Compared to only 3 keyframes in the proposed method, we show that our method incorporates better the behavior with keyframe constraints and provides more realistic and dynamic results.

6.4 User feedback

The prototype UI we implemented in Unity only includes a rough set of camera control features (select a style code, set and move keyframes with keyboard, select and generate trajectories). This showed to be insufficient to perform a thorough user evaluation that would be able to compare traditional keyframing techniques (for which many graphical gizmos and trajectory editing tools are available) with our technique. We therefore switched to an interview-driven approach with one senior previs artist and five senior art students in a film academy. First feedback from artists who saw or used the tool was that they appreciated the ability to use reference clips as an input to the system and one pointed out "it could be really interesting to analyse and compare the style of different directors". The senior previs artist add that "the tool could be good to the beginner in movie field since it provides an easy way to generate and edit camera trajectories" and that "short clips have become very popular and applying them to different scenarios has a great potential in production". He also insisted on the pedagogical value it could have by "using it in the film academies to teach how the cameras should move". To the question related to the quality of generated tracks, artists answer that they were mostly impressed by the cinematic value of the shots with so few keyframes. The zombie sequence was commented as "a great fighting scene with a first view camera which makes the zombie appear suddenly". The previs artist was more critique "the trajectories follow some recommendations and some times could give some very good shots". The mapping from reference clip to a new scene was sometimes difficult to comment on given the strong differences in content between the real sequence and the animation sequence. Artists then required a collection of additional features such as framing constraints to keep some characters in specific screen locations, attachment constraints or virtual targets (targets which are framed but not seen).

6.5 Playing with characteristic styles

To illustrate the capacity of our network to reproduce some "iconic" camera behaviors, we extracted style tags from chosen sequences which contain respectively a classical over-the-shoulder shot, and a low angle shot (camera is placed below the targets). Results displayed in Fig. 15 show how these iconic behaviors are reproduced in a synthetic sequence (see companion video for additional results).

Fig. 15. Two examples of "iconic" cinematic styles in real movies and generated results in our proposed system. First row displays on the left the reference shot with an identifiable over-the-shoulder framing, and second row displays a low-angle shot.

7 LIMITATION AND DISCUSSION

Artifacts for extreme keyframes. With the limitation of the intrinsic rules in camera behaviors, it is hard to simultaneously maintain the camera behaviors and the keyframes in all cases. Although our method tries to learn the balance between them, and despite adding a large amount of real data in the training to increase the possibilities in keyframes, when the keyframes are on extreme locations, the output of our method could not satisfy all the constraints.

One solution would be to make the balance between keyframe and style controllable, as a transition from extreme behaviors constraints to extreme keyframe constraints. An intuitive idea could be to control the ratio of linear interpolation to meet the keyframes from the generated curve. However, from the discussion in Section 5, interpolation would not ensure the implicit characteristics of the behavior. This requires a further exploration.

Camera behaviors limited to two characters. Though the proposed model is tailored to address scenes with two characters only, it can adapt to scenes with a single character by downsizing the input vector since the camera representation we rely on is a generalization of spherical coordinates for 1 to 2 targets (see Lino and Christie [2012]). For scenes with three targets or more, the framing problem is often over-constrained (see the P3P problem [Gao et al. 2003; Lino and Christie 2012]) if one does not use slanted camera angle, and can be trivially addressed by performing framing on the left-most and right-most characters as proposed in Galvane*et al.* [2013]. However, the very specific situations where characters enter or leave the frame is a path for future research.

8 CONCLUSION

In this paper, we propose a data-driven method for camera motion control. Given a 3D animation scene, our framework allows camera motion synthesis with a global similar camera behavior to a given reference film clip, and meanwhile satisfying local keyframe constraints specified by artist. In order to achieve this hybrid control strategy, we first designed a LSTM based gating network which can identify various camera behaviors embedded in the given reference clip and represent it as a latent style code. We further developed an autoregressive LSTM network for camera motion generation in which the high- and low- level behavior preferences are delivered into the system through the style code, the immediate next keyframe and its arriving time respectively. Moreover, we demonstrated in the paper that through hidden state mapping, the proposed LSTM camera motion prediction network can control camera behavior in an even finer manner.

This work represents the first to propose a camera in-betweening technique with style control through behavior specification. The work enables both very dense and sparse keyframes along the animation depending on the desired degree of control or automation. Future work will consist in extending the framework to handle more complex cinematic features as inputs (*e.g.* controlling the curvature of camera paths), more complex animated situations (characters entering or leaving the frame), and also consider the problem of cutting between multiple cameras under both high and low level constraints.

ACKNOWLEDGMENTS

We want to thank Anthony Mirabile and Yulong Zhang for the various support and helpful discussions throughout this project, as well as Yu Xiong for his help processing the MovieNet dataset. Furthermore, we wish to thank the anonymous reviewers for their constructive comments. This work was supported in part by the National Key R&D Program of China (2018YFB1403900, 2019YFF0302902).

REFERENCES

Ferran Argelaguet and Carlos Andujar. 2010. Automatic Speed Graph Generation for Predefined Camera Paths. In *Proceedings of the 10th International Conference on Smart Graphics*. Springer-Verlag, Berlin, Heidelberg, 115–126.

Jim Blinn. 1988. Where am I? What am I looking at?(cinematography). *IEEE Computer Graphics and Applications* 8, 4 (1988), 76–81.

Rogerio Bonatti, Arthur Bucker, Sebastian Scherer, Mustafa Mukadam, and Jessica Hodgins. 2020a. Batteries, camera, action! Learning a semantic control space for expressive robot cinematography. arXiv:2011.10118 [cs.CV]

Rogerio Bonatti, Wenshan Wang, Cherie Ho, Aayush Ahuja, Mirko Gschwindt, Efe Camci, Erdal Kayacan, Sanjiban Choudhury, and Sebastian Scherer. 2020b. Autonomous aerial cinematography in unstructured environments with learned artistic decision-making. *Journal of Field Robotics* 37, 4 (2020), 606–641.

Michael Büttner and Simon Clavet. 2015. Motion Matching-The Road to Next Gen Animation. *Proc. of Nucl. ai* (2015).

Loïc Ciccone, Cengiz Öztireli, and Robert W Sumner. 2019. Tangent-space optimization for interactive animation control. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–10.

Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. 2001. Composable Controllers for Physics-Based Character Animation. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. Association for Computing Machinery, New York, NY, USA, 251–260.

Quentin Galvane, Marc Christie, Christophe Lino, and Rémi Ronfard. 2015a. Camera-on-rails: automated computation of constrained camera paths. In *Proc. ACM SIGGRAPH Conf. Motion in Games*.

Quentin Galvane, Marc Christie, Rémi Ronfard, Chen-Kim Lim, and Marie-Paule Cani. 2013. Steering Behaviors for Autonomous Cameras. In *Proceedings of Motion on Games*. Association for Computing Machinery, New York, NY, USA, 93–102.

Quentin Galvane, Rémi Ronfard, Christophe Lino, and Marc Christie. 2015b. Continuity Editing for 3D Animation. *Proceedings of the AAAI Conference on Artificial Intelligence* 29, 1 (Feb. 2015).

Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. 2003. Complete solution classification for the perspective-three-point problem. *IEEE transactions on pattern analysis and machine intelligence* 25, 8 (2003), 930–943.

Mirko Gschwindt, Efe Camci, Rogerio Bonatti, Wenshan Wang, Erdal Kayacan, and Sebastian Scherer. 2019. Can a Robot Become a Movie Director? Learning Artistic Principles for Aerial Cinematography. arXiv:1904.02579 [cs.RO]

Félix G Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020. Robust motion in-betweening. *ACM Trans. on Graphics* 39, 4 (2020), 60–1.

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

Axel Hösl. 2019. *Understanding and designing for control in camera operation.* Ph.D. Dissertation. lmu.

Chong Huang, Yuanjie Dang, Peng Chen, Xin Yang, et al. 2019a. One-Shot Imitation Filming of Human Motion Videos. *arXiv preprint arXiv:1912.10609* (2019).

Chong Huang, Chuan-En Lin, Zhenyu Yang, Yan Kong, Peng Chen, Xin Yang, and Kwang-Ting Cheng. 2019b. Learning to film from professional human motion videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4244–4253.

Chong Huang, Zhenyu Yang, Yan Kong, Peng Chen, Xin Yang, and Kwang-Ting Tim Cheng. 2019c. Learning to capture a film-look video with a camera drone. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 1871–1877.

H. Huang, D. Lischinski, Z. Hao, M. Gong, M. Christie, and D. Cohen-Or. 2016. Trip Synopsis: 60km in 60sec. *Computer Graphics Forum* 35, 7 (2016), 107–116.

Qingqiu Huang, Yu Xiong, Anyi Rao, Jiaze Wang, and Dahua Lin. 2020. MovieNet: A Holistic Dataset for Movie Understanding. In *Proc. European Conference on Computer Vision*.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation* 3, 1 (1991), 79–87.

Hongda Jiang, Bin Wang, Xi Wang, Marc Christie, and Baoquan Chen. 2020. Example-Driven Virtual Cinematography by Learning Camera Behaviors. *ACM Trans. Graph.* 39, 4 (2020), 14.

Diederick P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Int'l Conf. Learning Representations*.

C. Lino and M. Christie. 2012. Efficient Composition for Virtual Camera Control. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Computer Animation*.
Christophe Lino and Marc Christie. 2015. Intuitive and Efficient Camera Control with the Toric Space. *ACM Trans. Graph.* 34, 4, Article 82 (2015), 12 pages.
Eric Marchand and Nicolas Courty. 2000. Image-based virtual camera motion strategies. In *Graphics Interface Conference, GI'00*. Morgan Kaufmann, 69–76.
Jianyuan Min and Jinxiang Chai. 2012. Motion Graphs++: A Compact Generative Model for Semantic Motion Analysis and Synthesis. *ACM Trans. Graph.* 31, 6, Article 153 (November 2012), 12 pages.
Patrick Olivier, Nicolas Halper, Jon Pickering, and Pamela Luna. 1999. Visual composition as optimisation. In *AISB symposium on AI and creativity in entertainment and visual art*, Vol. 1. 22–30.
Thomas Oskam, Robert W. Sumner, Nils Thuerey, and Markus Gross. 2009. Visibility Transition Planning for Dynamic Camera Control. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Association for Computing Machinery, New York, NY, USA, 55–65.
Andrew Witkin and Michael Kass. 1988. Spacetime Constraints. *SIGGRAPH Comput. Graph.* 22, 4 (1988), 159–168.
Xinyi Zhang and Michiel van de Panne. 2018. Data-Driven Autocompletion for Keyframe Animation. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*. Association for Computing Machinery, New York, NY, USA, Article 10, 11 pages.

A APPENDIX

A.1 NETWORK STRUCTURE

The full architecture of our network is summarized in the table below, where FC denote Fully-connected layers. The number of input and output channels are reported in the rightmost column.

Name	Layers	in/out
Gating Network	LSTM	$60 * 14/256$
	FC	256/4
Prediction Network	FC + ReLU	160/256
	FC + ReLU	256/256
	LSTM	256/256
	FC + ReLU	256/256
	FC	256/5
Hidden Mapping	FC + ReLU	29/256
	FC	$256/256 * 2$