Example-driven Virtual Cinematography by Learning Camera Behaviors

HONGDA JIANG*, CFCS, Peking University & AICFVE, Beijing Film Academy
BIN WANG*, AICFVE, Beijing Film Academy
XI WANG, University Rennes, Inria, CNRS, IRISA & AICFVE, Beijing Film Academy
MARC CHRISTIE, University Rennes, Inria, CNRS, IRISA & AICFVE, Beijing Film Academy
BAOQUAN CHEN†, CFCS, Peking University & AICFVE, Beijing Film Academy

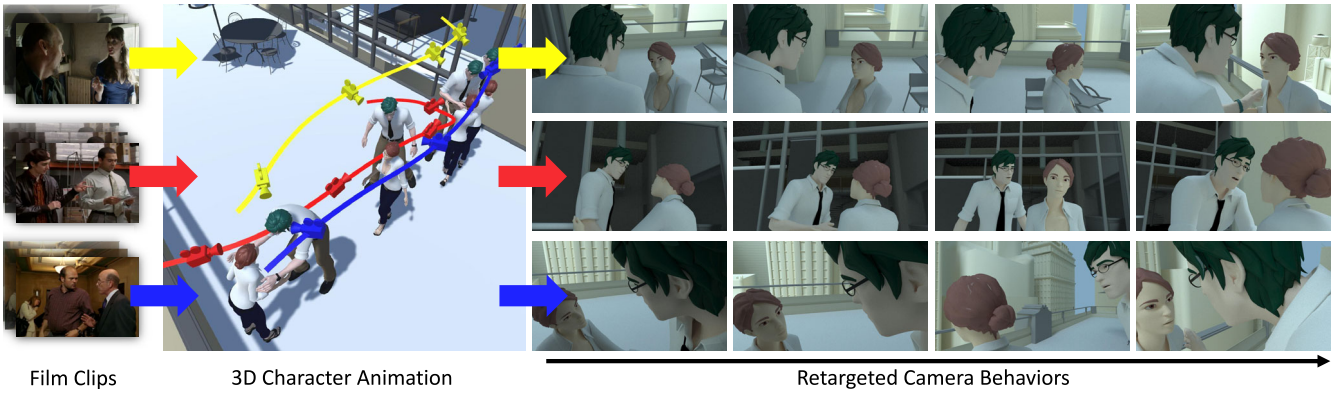

Fig. 1. We propose the design of a camera motion controller which has the ability to automatically extract camera behaviors from different film clips (on the left) and re-apply these behaviors to a 3D animation (center). In this example, three distinct camera trajectories are automatically generated (red, blue and yellow curves) from three different reference clips. Results display viewpoints at 4 specific instants along each camera trajectory demonstrating the capacity of our system to encode and reproduce camera behaviors from distinct input examples.

Designing a camera motion controller that has the capacity to move a virtual camera automatically in relation with contents of a 3D animation, in a cinematographic and principled way, is a complex and challenging task. Many cinematographic rules exist, yet practice shows there are significant stylistic variations in how these can be applied.

In this paper, we propose an example-driven camera controller which can extract camera behaviors from an example film clip and re-apply the extracted behaviors to a 3D animation, through learning from a collection of camera motions. Our first technical contribution is the design of a low-dimensional cinematic feature space that captures the essence of a film's cinematic characteristics (camera angle and distance, screen composition and character configurations) and which is coupled with a neural network to automatically extract these cinematic characteristics from real film clips. Our second technical contribution is the design of a cascaded deep-learning architecture trained to (i) recognize a variety of camera motion behaviors from the extracted cinematic features, and (ii) predict the future motion of a virtual camera given a character 3D animation. We propose to rely on a Mixture of Experts (MoE) gating+prediction mechanism to ensure that distinct camera behaviors can be learned while ensuring generalization.

We demonstrate the features of our approach through experiments that highlight (i) the quality of our cinematic feature extractor (ii) the capacity to learn a range of behaviors through the gating mechanism, and (iii) the ability to generate a variety of camera motions by applying different behaviors extracted from film clips. Such an example-driven approach offers a high level of controllability which opens new possibilities toward a deeper understanding of cinematographic style and enhanced possibilities in exploiting real film data in virtual environments.

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

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

ACM Reference Format:
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, Article 45 (July 2020), 14 pages. https://doi.org/10.1145/3386569.3392427

*denotes equal contribution
†corresponding author

Authors' addresses: Hongda Jiang, jianghd@pku.edu.cn, CFCS, Peking University & AICFVE, Beijing Film Academy; Bin Wang, binwangbuaa@gmail.com, AICFVE, Beijing Film Academy; Xi Wang, xi.wang@inria.fr, University Rennes, Inria, CNRS, IRISA & AICFVE, Beijing Film Academy; Marc Christie, marc.christie@irisa.fr, University Rennes, Inria, CNRS, IRISA & AICFVE, Beijing Film Academy; Baoquan Chen, baoquan@pku.edu.cn, CFCS, Peking University & AICFVE, Beijing Film Academy.

0730-0301/2020/7-ART45 $15.00
https://doi.org/10.1145/3386569.3392427

1 INTRODUCTION

Virtual cinematography – the process of placing and moving virtual cameras through three dimensional environments – finds many applications in the movie and game industries. However, formalizing filming techniques into computational models remains a challenge. While there are many rules which guide how events should be portrayed by cameras [Arijon 1991], significant variations are encountered in how these should be applied, making the design of a general purpose camera motion controller a challenging task to achieve.

Partial answers have been proposed: retargeting low and high frequency signals from real camera trajectories to virtual environments [Kurz et al. 2014]; designing dedicated camera steering behaviors using physical models [Galvane et al. 2013]; performing trajectory optimization given waypoint constraints [Huang et al. 2016] or designing specific camera interpolators defined in alternate camera spaces, the benefit of which is to maintain visual features over time (framing, camera angle, distance) [Galvane et al. 2015a].

An attractive alternative would be *to train a camera motion controller* from real footage in a way to encode the correlations between the camera motion and the scene contents (in the following we will refer to these temporal correlations as *camera motion behaviors*). The benefit of training is that these cinematographic behaviors are implicitly encoded rather than explicitly implemented. While there are no such approaches in computer animation, multiple contributions in the field of drone cinematography have explored how drone motions could be learned by extracting relative camera angles from real footage, and designing a camera motion predictor that relies on supervised learning [Huang et al. 2019b,c] or specific reinforcement learning [Bonatti et al. 2019; Passalis and Tefas 2019]. Such approaches however focus on a problem limited to shooting one target only, either from a long distance aerial view or a front close-up shooting [Bonatti et al. 2019]. Proposed methods are also hampered by the necessity of manually annotating film footage to handle different types of camera motions; the results, though quite distinct, lack variety and users have a rather limited degree of controllability on the generated sequence (a choice of one style among n).

In this paper we propose the design of an *example-driven camera motion controller* for computer animation which (i) is able to handle more general situations than one-target-only drone cinematography, specifically two-character interactions which are commonplace in movies, (ii) is trained with a range of different camera behaviors taken from synthetic and real movie clips and (iii) has the ability to automatically extract a sequence of camera behaviors from a user-selected film clip (or concatenated pieces of clips) and to retarget them to a virtual environment.

A first requirement is to design a low-dimensional *Cinematic Feature Space* as a mathematical model representing the camera's spatial configuration in relation with two characters. The feature space we propose captures the essential cinematic characteristics of an image (camera angles, distance to characters, on-screen layout and characters relative configurations) and has the ability to retarget these characteristics on a $3D$ animation using a relative camera representation for cinematic camera placement (Toric space [Lino and Christie 2015]). Building on this representation, we propose the first practical tool to automatically analyze and extract sequences of cinematic features from example film clips through the provision of a deep learning *Cinematic Feature Estimator*.

A second requirement is the design of a *camera motion prediction network* able to predict the next camera poses given a sequence of past cinematic features and a 3D animation. Training such a network may seem intractable because of the inherent ambiguity in camera/character motion correlations. Indeed, significantly different camera motions (behaviors) can occur for the same character animation. A parallel can be drawn with the design of character motion controllers that need to account for different locomotion styles. Different solutions have been proposed in this context including residual adapters [Mason et al. 2018], hot-style vectors for annotated data [Smith et al. 2019] or Mixture of Experts [Zhang et al. 2018]. Inspired by the latter, we employ a Mixture of Experts (MoE) scheme in which multiple experts (*i.e.* different prediction networks) are coordinated with a gating network whose role is to decide which experts to activate. Each expert gets specialized on a different region of the input data (*i.e.* different camera behavior). The strength of such an approach comes from that experts and gating networks are trained jointly by back propagation, hence imposing no requirement on manual pre-processing or labelling of the dataset.

A key feature of this approach is that the gating network acts as an encoder, hence characterizing a low dimension manifold of valid camera behaviors: a *Camera Behavior Space*. Once trained, the network has the ability to identify from a reference clip, the corresponding sequence in the behavior space, and can therefore apply the appropriate combination of experts to reproduce this sequence in a new 3D animation. The approach is limited to encoding and reproducing continuous motions of the camera and therefore does not handle cuts between different camera angles.

In summary the contributions of our paper are:

- a low-dimensional cinematic feature space which can be used both to encode cinematic characteristics of a real film clip, and retarget these characteristics to a 3D animation;
- a novel deep learning cinematic feature estimator that is trained to automatically extract features from real film footage and express them in the cinematic feature space.
- a deep learning gating+predicting network designed to learn camera behaviors from correlations in the cinematic feature space *without any style labelling* and on a small set of camera behaviors (4 in our case);
- a full fledged camera motion controller which, given an example film clip (or a sequence of clips) is able to identify the camera behaviors, and retarget them to a 3D animation, offering a high-level controllable way of creating camera motions.

To the best of our knowledge, this is the very first example-driven camera motion controller for virtual cinematography, able to encode and reproduce continuous behaviors of cameras in relation to two characters.

2 RELATED WORK

Camera planning in virtual environments is an active research topic [Christie et al. 2008] and despite some long term cross-cutting

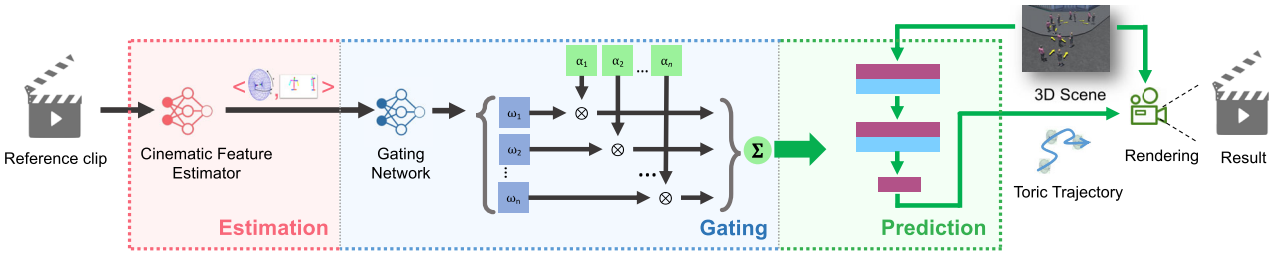

Fig. 2. Our proposed framework for learning camera behaviors composed of a Cinematic Feature Estimator which extracts high-level features from movie clips, a Gating network which estimates the type of camera behavior from the high-level features, and a Prediction network which applies the estimated behavior on a 3D animation.

issues such as viewpoint quality estimation, proposed solutions are dedicated to address different classes of problems (real-time tracking of targets [Halper et al. 2001; Oskam et al. 2009], automated shot and edit planning [Galvane et al. 2015b; Ranon and Urli 2014] or designing cinematic narrative experiences [Galvane et al. 2014]).

Different techniques have been proposed to automatically place a camera using different levels of specification. General approaches consist in expressing visual properties as a composition of cost functions applied to the camera parameters, and using constraint solving [Bares et al. 2000], numerical optimization [Drucker et al. 1992; Ranon and Urli 2014], or hybrid numerical/geometric [Lino et al. 2010] optimization. The generality of these approaches is counterbalanced by significant computational costs.

2.1 Camera Motion Planning

In comparison, the problem of computing camera motions has received less attention. Different path or motion-planning techniques have been proposed to guide the design of camera trajectories mostly inspired by the robotics literature (see [Lino et al. 2010; Oskam et al. 2009]). Optimization has also been considered to solve high-level requirements, addressing the placement of way-points [Huang et al. 2016], or parameters of curve representations [Galvane et al. 2015a]. The key issue common to most camera planning approaches is actually how to characterize what makes a good motion.

Rather than trying to formalize the characteristics of such motions, multiple contributions have relied on a data-driven approach that exploit camera paths extracted from real film clips. In [Kurz et al. 2014], the authors relied on Structure from Motion (SfM) to extract camera paths and characterize their spectral properties using low frequency and high frequency features in a way to re-apply the motion from the film clip to the virtual environments (motion style transfer). Camera motion styles are manually labeled and then queried by the user. The computed paths however do not adapt to the contents of the 3D scene (no collision, occlusion avoidance or framing of any targets). Inspired by the former approach Sanokho *etal.* [Sanokho et al. 2014] also adopted a data-driven approach for real-time cinematography in games by extracting camera paths using SfM, converting these paths in Toric space coordinates [Lino and Christie 2015] using manual annotation of on-screen targets and then building *a camera motion graph* that expresses all extracted paths in a joint basis and connecting the paths with transitions and cuts. At run time, the camera system would choose which path to follow, and when to cut between different paths. The system however does not learn the behavior of the cameras, it simply selects the motions in the graph which avoid the occlusion of the targets. Another relevant way to characterize camera motion is to study the correlation between the motion of the camera and the changes in the 3D scene [Assa et al. 2010], *i.e.* study the behavior of the camera. By transposing Reynolds' work on steering behaviors [Reynolds 1999], Galvane *et al.* [Galvane et al. 2013] designed of a range of *camera steering behaviors*, each triggered by events occurring in the 3D scene and by the evolution in character configurations (*e.g.* characters grouping, characters splitting, *etc.*). The approach however remains limited by the range of behaviors that need to be implemented and the difficulty in deciding priorities between the behaviors or the impossibility to blend behaviors. In our approach, we rather propose to extract behaviors from real-film clips, using a deep learning network trained on a limited set of behaviors.

2.2 Learning-based Camera Control

Research in drone cinematography started to explore the use of learning from real footage (also called imitation filming) to drive the motion of drones. By relying on a deep reinforcement learning approach [Gschwindt et al. 2019], drones displayed the capacity to automatically select the type of shot (one angle among four) that maximises a reward based on aesthetics, by estimating the target configuration in real-time. This work is further extended by [Bonatti et al. 2019] to accommodate for occlusion avoidance and performing trajectory optimization. Passalis et al. [2019] improved the reinforcement learning convergence speed and accuracy by designing a novel reward function. In [Huang et al. 2019c], the authors train a prediction network from drone video clips (using a sliding window over each clip). Clips are analyzed with YOLOv3 and OpenPose to extract the character's skeletal information (the screen height and relative position *w.r.t.* camera). The trained prediction network is then used to predict the next location of the drone, given the past locations and current pose of the character. Huang et al. [2019b] further extended the work by involving background's dense optical flow in the network to get a better sense of the relative motion between the character and the background. A one-shot learning technique has recently been proposed in [Huang et al. 2019a] using a style feature extraction network to detect style from a reference drone video and apply it to a new sequence. Our work relies on a similar

pipeline. However, in comparison, we propose a more general approach (dealing with more complex shot compositions, shot sizes and camera angles, while also featuring shots with two characters) by explicitly relying on a gating network to encode different types of camera behaviors in a way to avoid prior labelling.

3 OVERVIEW

Following the objective of designing a camera motion controller which can learn camera behaviors and transfer extracted behaviors from example clips to a 3D animation, we designed a three-staged pipeline illustrated in Fig. 2.

The pipeline takes as input a real film clip (or a sequence of clips) which serves as reference, a 3D animation with characters, and outputs a camera motion. We use Mixture of Experts as a backbone structure and the pipeline is composed of:

(i) a *cinematic feature estimator* which extracts relevant features from film clips (as input) and expresses them in a cinematic feature space (as output)

(ii) a *gating network* which acts as a behavior selector through the construction of a low dimensional manifold of learned camera behaviors (our latent camera behavior space). The input of the trained gating network is a path in the cinematic feature space, and the output is a sequence of camera behaviors which will then be used as expert weights to specialize the camera prediction network.

(iii) a *camera pose prediction network* in which experts, activated by the gating network, can predict the next camera pose given a sliding window of camera and character poses from a 3D animation. The output of the prediction network is a relative camera pose expressed in Toric space coordinates (a camera representation which encodes cinematic principles [Lino and Christie 2015]) which can then be applied to a 3D animation to compute the final camera pose.

This paper is organized as follows: we start by presenting the cinematic feature space with the identification of relevant feature for learning and detail the design of our cinematic feature extractor (Section 4). We then describe the camera motion controller in Section 5 and evaluate its individual components in Section 6. Results are reported in Section 7 before discussing limitations (see Section 8) and concluding in Section 9.

4 CINEMATIC FEATURE ESTIMATOR

In this paper, we propose a low-dimensional space: the *cinematic feature space*, to describe characteristic relations between camera and characters. The space is designed in mind to (i) ease the extraction of cinematic features from real footage and (ii) enable the reconstruction of a camera pose in 3D environment from the cinematic features. The representation includes camera and character relative positions and orientations, as well as 2D framing information, hence capturing the essence of the cinematic properties that compose a shot [Arijon 1991].

Based on this representation we also designed a Cinematic Feature Estimator, which extracts cinematic features from RGB images with characters. The feature estimation pipeline is presented in Fig. 3. It is composed of three stages (i) a character pose estimation stage based on LCR-Net [Rogez et al. 2019] to extract 2D character poses from images, (ii) a pose association and filling stage to match characters over different frames and improve robustness in pose detection through temporal coherency, and (iii) a neural network trained to output the cinematic features from temporally coherent character poses. The first and second stages are designed to reduce the dimension of the learning problem, which eases the convergence and makes it less data quantity demanding. Since these two parts are not the contribution of this paper, readers are encouraged to refer to Appendix B for details.

4.1 Cinematic Feature Space

Spatial relations (distances and angles) between characters as well as camera framings (how characters are composed in the screen, under which camera angle and distance) are key factors for storytelling in film media. Coordinating them in different ways can convey distinctive messages to the audience, and compose the essence of a *camera behavior*. The decision as to which features should be extracted from film clips is guided by the need to understand these behaviors,*i.e.* understanding the correlations between the evolving configurations of two characters and evolution of camera angles, framing and distances.

Camera features. Estimating a camera pose from a single RGB image remains an open challenge; solutions can only rely on strong priors on the content of the scene (*e.g.* Manhattan assumption). Our work builds on the hypothesis that at least two characters are present on screen, and that a camera relative pose can be estimated from these character poses by training a dedicated network. In this work we propose to express our camera pose in the Toric space coordinate system [Lino and Christie 2015], an expressive and compact local representation dedicated to camera framing, manipulation and interpolation tasks based on two given targets (see Fig. 4(c)). Note that the process can be easily generalized to work with a single character as in [Huang et al. 2019b] using polar coordinates, the Toric space already being a generalisation of a polar coordinate system to two targets. By definition, a camera feature $\mathbf{c}$, for a given field of view and two targets A and B is expressed in the Toric space using the following representation:

$$\mathbf{c} = \{\mathbf{p}_A, \mathbf{p}_B, \theta, \varphi\} \in \mathbb{R}^6$$

where $\mathbf{p}_A$ and $\mathbf{p}_B$ represent the screen positions of each character. θ and φ are two parametric angles, intuitively representing the yaw and pitch angles *w.r.t.* to the line AB defined between the 3D characters. In this representation, given two desired screen positions, the set of possible camera viewpoints formulate a 2-parametric manifold surface (θ, φ) shaped like a spindle torus, on which every viewpoint enforces the same angle α between the two targets (computed from the screen positions) and the same visual composition (on-screen positions of A and B). The benefit of this representation in our context is three-fold (i) the camera being defined in the local bases of targets A and B, one can easily correlate the 2D on-screen motion of characters with the relative motion of the camera in Toric coordinates, (ii) the on-screen position of the characters is embedded in the model, making it easy to re-apply the framing to different 3D environments and (iii) the set of parameters are easy to learn due to the strong correlation between the input (on-screen character poses) and output (camera pose in Toric coordinates).

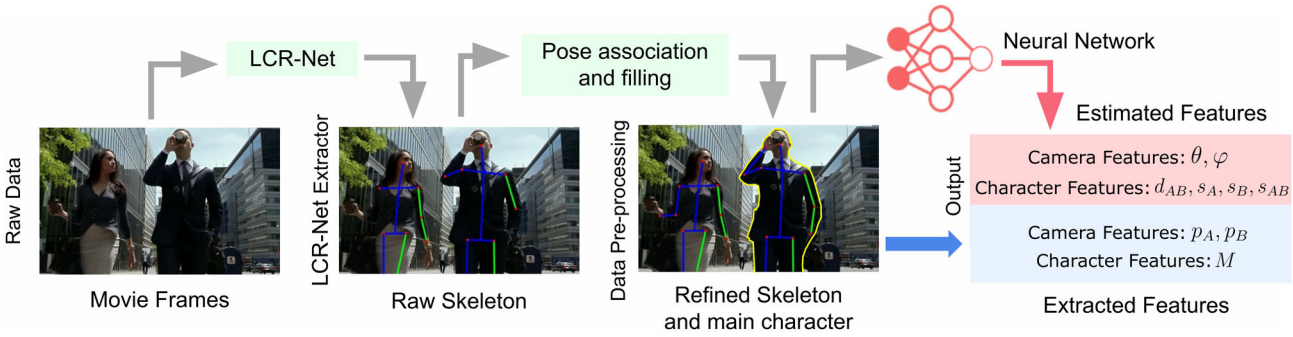

Fig. 3. To estimate cinematic features from film clips, each frame should pass through the following steps: (i) extracting 2D skeletons with LCR-Net, (ii) pose association, filling missing joints and smoothing, and (iii) estimating features through a neural network. Camera features θ, φ and character features d_{AB}, s_A, s_B, s_{AB} are generated by the neural network estimator (see Fig. 5), whilst position of actors heads $\mathbf{p}_A, \mathbf{p}_B$ as well as main character information M are obtained directly from the result of previous stages.

Character features. The design of the cinematic character features is guided by (i) the possibility to learn such variables from the 2D pose estimations and (ii) the capacity of these variables to enable the gating+prediction network to learn camera behaviors. Different geometric quantities were tested (elbow positions, head orientations, difference between head/shoulder angles, sizes) and the following set of variables was finally retained:

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

As illustrated in Fig. 4(b), d_{AB} represents the 3D distance between two characters, s_A (resp. s_B) represents the angle between line AB and the front vector orthogonal to the segment linking the shoulders of character A (resp. B). Angle s_{AB} represents the difference between the shoulders orientations of the two characters. M is a binary-valued variable indicating who is the main character of the sequence (see Appendix B for details).

Feature extraction. The extraction of screen composition values $\mathbf{p}_A$ and $\mathbf{p}_B$ is straightforward: values are provided by the head position through the character pose estimation. The *main character* feature M is also estimated from the poses, by using Hitchcock's cinematographic principle: the most important should be the largest displayed in the screen over a sequence (see details in Appendix B). The feature variables we want to learn using neural network are therefore the camera pose in Toric coordinates defined by (θ, φ) and character relative position $(d_{AB}, s_A, s_B, s_{AB})$.

4.2 Network Structure

The structure of our estimator network is illustrated in Fig. 5. We use as input the character joints from temporally corrected LCR-Net data. Instead of using all of them, we select joints which provide a hint on position, size and orientation of characters *w.r.t.* camera, and also for their relative accurate position estimation in LCR-Net. Thus, as illustrated in Fig. 4(c), the input feature vector $\mathbf{u} \in \mathbb{R}^{28}$ for each frame is defined as:

$$\mathbf{u} = \{\mathbf{p}_A, \mathbf{p}_B, \mathbf{n}_A, \mathbf{n}_B, \mathbf{h}_A, \mathbf{h}_B\} \quad (2)$$

where:

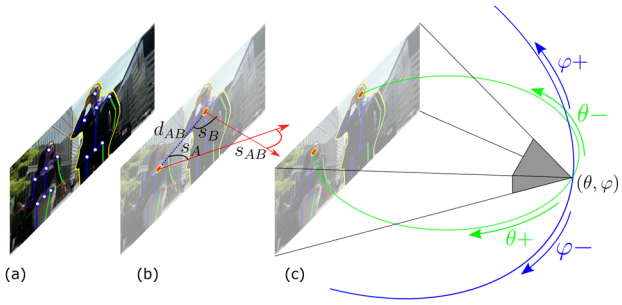

Fig. 4. Estimation of cinematic features. From left to right, (a) the LCR-Net pose detection with selected 2D joints, (b) the estimated character features such as inter-character distance, absolute and relative orientations on yaw direction of shoulders and (c) the estimated camera pose expressed in relative Toric space coordinates that encodes a camera pose using framing features (the 2D on-screen positions of characters) together with pitch and yaw angles (θ, φ).

- vectors $\mathbf{p}_A, \mathbf{p}_B \in \mathbb{R}^2$ represent the on-screen 2D head joint positions of the two characters (A being the character on the left of the first frame of the sequence); Head positions are generally power-points in image composition [Mascelli 1965] *i.e.* points to which our gaze is naturally attracted.
- vectors $\mathbf{n}_A, \mathbf{n}_B \in \mathbb{R}^6$ represent neck, left and right shoulder joint positions of character A and B, giving an idea of the relative orientations between the characters, and in relation to the camera;
- vectors $\mathbf{h}_A, \mathbf{h}_B \in \mathbb{R}^6$ represent the hip, left and right upper leg joint positions that should provide a hint on the projected vertical size of the characters, hence on the distance of the camera to the characters, and on the camera's tilt angle.

Leveraging on the temporal consistency in film clips, we rely on a sliding window technique to handle the possible noise in the input data and improve robustness. A window size of $t = 8$ frames is empirically chosen for this network which contains 4 previous and 3 following consecutive frames around the target frame i as:

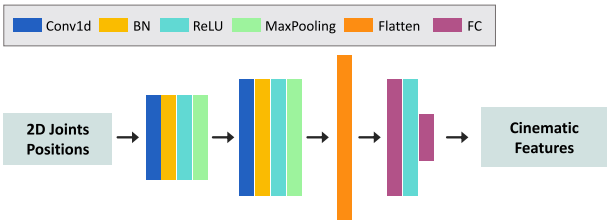

Fig. 5. Learning stage of the Cinematic Feature Estimator. Input data is gathered over on a sliding window of $t_c = 8$ frames to increase robustness during the learning and testing phases. The output data gathers cinematic features such as the camera pose estimation in Toric space (θ, φ), the distance between characters d_{AB} and relative shoulder orientations (s_A, s_B, s_{AB}).

$\mathbf{x}_i = \{\mathbf{u}_{i-4}, \mathbf{u}_{i-3}, ...\mathbf{u}_i, \mathbf{u}_{i+1}, ...\mathbf{u}_{i+3}\} \in \mathbb{R}^{8\times 28}$. A one dimensional convolution is performed on input data over the temporal domain, independently for each joint channel. The corresponding output of the estimation network at frame i is therefore defined as:

$$\mathbf{y}_i = \{\theta, \varphi, d_{AB}, s_A, s_B, s_{AB}\} \in \mathbb{R}^6 \tag{3}$$

In order to maintain the generality of our network with moderate amount of training data, we normalized the input in the following way (i) we know that the absolute on-screen positions of the two characters are independent from angles φ and θ (see [Lino and Christie 2015], only the horizontal and vertical differences between on-screen positions influence φ and θ); (ii) we can shift the two characters in image space horizontally or vertically while conserving their on-screen distance and (iii) this shift can be approximately interpreted as a pure camera rotation with an small upper bound error related to the camera's field of view (FOV), which is commonly no more than 45° in real movie shooting (for a detailed derivation, please refer to Appendix E). Based on the above observations, these 6 parameters in $\mathbf{y}_i$ are estimated separately using three different networks with the same structure, but different weights and inputs: (i) we first translate two characters on the screen vertically to place the center of $\mathbf{p}_A$ and $\mathbf{p}_B$ at the center of screen along Y direction to derive θ. (ii) similarly the 2D poses centered along X direction are then used to estimate camera pitch angle (φ); (iii) finally, character features d_{AB}, s_A, s_B and s_{AB} are evaluated from the 2D pose centered on both X and Y direction. For a detailed description of the parameters of each layer, please refer to the Appendix A.

4.3 Data Preparation and Training

We propose to train the feature estimator network by only using synthetic data (input is 2D joint positions). We start by selecting a range of 3D character animations, and then generate pieces of camera sequences (8 frames) in which we perform all possible variations on Toric parameters φ, θ and $\mathbf{p_A}, \mathbf{p_B}$ using a range of sinusoidal functions with different frequencies and amplitudes as:

$$\mathscr{F}(t) = k_\Gamma \cdot \Gamma \cdot \sin(k_\Omega \cdot t) + \Psi \tag{4}$$

where Γ and Ψ are the amplitude and median value of each variable, k_Γ and $k_\Omega \sim \mathcal{U}(0, 0.01^2)$. Given the small size of the sliding window, this provides a mean to express a plausible range of variations in camera parameters. Due to the general continuous nature of camera paths, there is no need to generate high-frequency variations. We augment the data by inserting random noise (≤ 10 deg) directly on characters' 3D joint angles.

For each data sample (8 frames with a variation on the character motion driven by the animation and a variation on the camera Toric space parameters), we use the 3D animation skeleton joints for the characters (selecting the joints which correspond to the LCR-Net joints) and perform the on-screen projection. All 2D joint coordinates are normalized into on-screen positions in the $[0, 1]$ range, both horizontally and vertically to remain independent of the screen resolution. All the training data are generated with the same aspect ratio and FOV. For testing with different FOV or aspect ratios, a straightforward linear transformation is used to align the input 2d on-screen joint position *w.r.t.* perspective projection.

Since we address a typical regression task, we train our network using following loss function:

$$\mathscr{L}(\tilde{\mathbf{y}}, \bar{\mathbf{y}}) = \|\tilde{\mathbf{y}} - \bar{\mathbf{y}}\| \tag{5}$$

which is the mean of squared differences between the desired output $\bar{\mathbf{y}}$ and the output of network $\tilde{\mathbf{y}}$ for a given training sample x_i. $\bar{\mathbf{y}}$ is achieved by normalizing $\mathbf{y}$ on each individual channel.

Our method is trained using the Adam adaptive gradient descent algorithm [Kingma and Ba 2015] on a dataset with 770000 annotated data, generated from 30 different animation sequences. Training is performed for 100 epochs and takes around 50 minutes on a NVIDIA GeForce Titan Xp GPU with batch size of 256. We use the exponential learning rate policy with base learning rate set to 0.001 and decay 0.95 after each epoch.

5 CAMERA MOTION CONTROLLER

Studying correlations between camera and character motions is a typical time series modeling problem. Given rich time series data, a prediction network can be easily designed to learn and predict the future poses of a camera in response to the past and current states of the camera and characters (as displayed in the right part of Fig. 6). However, there is a challenging issue because of the inherent ambiguity of real cinematography where there exists a large variety of camera behaviors: in essence for a very similar input sequence (past and current states of characters and camera), different camera motions can occur and there is no rule to simply determine the best behavior from solely input information. While modern feed-forward neural networks have proven extremely effective in various inference and classification tasks, they tend to regress toward mean values if there are such ambiguities in the training data.

Inspired by results on the development of a quadruped motion controller [Zhang et al. 2018], we rely on a Mixture of Experts (MoE) algorithm [Jacobs et al. 1991] to solve the issue of learning and embedding multiple camera behaviors in the same network, without performing prior behavior labelling on the input data. MoE uses multiple experts to divide the space of original problem (training data) into several homogeneous regions and trains each expert to only specialize in a subset of the training cases. As illustrated in Fig. 6, the entire pipeline is divided into two parts: gating network and prediction network.

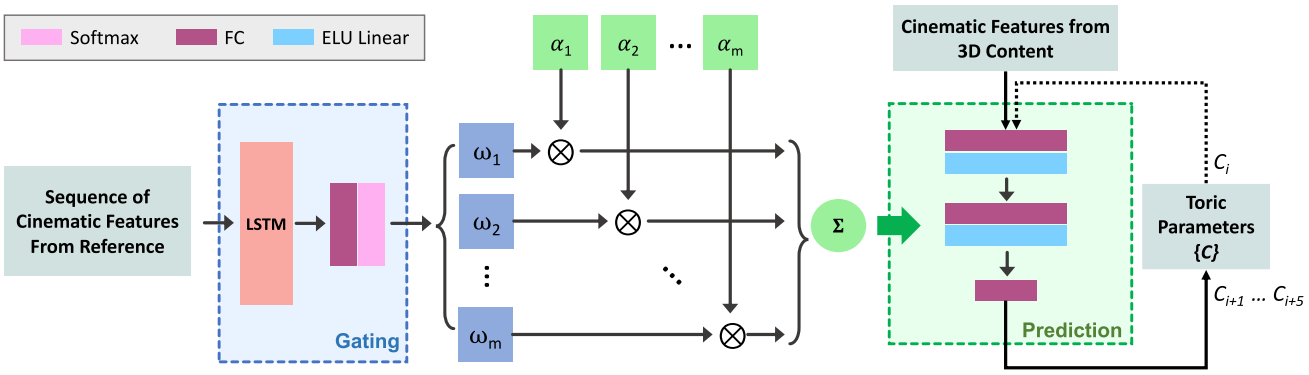

Fig. 6. Structure of our Mixture of Experts (MoE) training network. The network takes as input the result of the Cinematic Feature Estimator applied on reference clips and the 3D animation. It outputs a sequence of camera parameters for each frame i of the animation that can be used to render the animation. Training is performed simultaneously at different time scales: the gating network is trained to decide which experts to activate by using a LSTM analyzing longer sequences (t_g = 400 frames), and the prediction network is trained to reproduce behaviors from shorter input sequences (t_p = 120 frames).

5.1 Gating Network

The gating network is responsible for extracting the specific behavior (or combination of behaviors) in a given reference clip and then activate specific experts in the prediction network by weighing their relative influence. The architecture of the gating network is described in the left part of Fig. 6. Gating network input is a sequence of cinematic features from the reference clip. The global latent camera behavior information of the input sequence is captured through a standard single-output LSTM layer, and further compressed to a dimension m vector using a fully connected layer. A softmax operator at the end of our gating network normalizes the m dimension behavior feature such that they sum up to 1, which is required for further linear blending.

According to our tests, short sequences of cinematic features cannot provide reliable enough information to distinguish different behaviors, and we therefore exploit sequences of at least 400 frames for all the reference clips. The meta value m represents the number experts used in our system, which needs to be adjusted according to the complexity of the training data. The m-dimensional vector can also be regarded as a latent style label of the given reference clip. For more discussions please refer to Section 6.

5.2 Prediction Network

The prediction network is in charge of computing the camera motion given a 3D animation and weights from the gating network. As illustrated in the right part of Fig. 6, the prediction network is only a simple three FC-layered neural network. The network takes as input character cinematic features from a 3D animation, a sliding window centered at frame i and also camera poses from past frames to predict new camera poses in the near future. The global weights of the prediction network are computed by blending m trained coefficients $\alpha_1, ..., \alpha_m$ where $\alpha = \sum \omega_i \alpha_i$ and $\omega = \omega_1, ..., \omega_m$ are the weights generated by the gating network.

Compared with the gating network, our prediction network has a relatively small time horizon. In our implementation, we set the local window centered at frame i, containing 60 frames in past and 59 frames in the future. The input contains character cinematic features for all 120 frames and camera features for only past 60 frames; the input of the trained network has size $60 * 5 + 120 * 9 = 1380$. The network outputs camera Toric parameters for the future 30 frames with total dimension as $30 * 5 = 150$. We have also conducted a test by reducing the output to only 1 frame, and found that the change of camera pose between two consecutive frames is too subtle to force the network to use all 3D content information to perform a reasonable prediction.

5.3 Data Preparation and Training

We create a hybrid dataset to train our gating+prediction network in which 90% of the data is synthetic data (30 existing 3D animated scenes with length of 1500 frames, either created by key-framing or by motion capture animation), while the other 10% comes from real film footage (62 real film clips) on which the cinematic feature estimator has been applied. When generating the synthetic data, we implement four well known cinematography behaviors taken from film clips (direct track, side track, relative track and orbiting, see companion video and Appendix D for more explanation), through simple parametric models and compute variations of these behaviors through a regular sampling of the behaviors parameters. Among these four behaviors, direct track and relative track are tightly coupled with the characters relative configurations. The other two motions can be controlled by high-level parameters (*e.g.* frequency in the case of the orbit behavior). For each of these behaviors, we also add variations in terms of shot size (how large characters appear on the screen). We consider three shot types (close-up, medium shot, and long shot) which we enforced by simply changing the distance between the projected positions of heads $\mathbf{p}_A$ and $\mathbf{p}_B$ (hence moving the camera closer or further).

Since gating and prediction networks have different temporal windows size, we take cinematic features of 400 consecutive frames at arbitrary positions in the input sequences and camera Toric parameters of another 90 consecutive frames (60 for input and 30 for output) from the same sequence to formulate a training data sample. We have 2, 160, 000 data samples in total. For a detailed description

of the parameters of each layer, please refer to the Appendix A. We augment the prediction input data in two ways: (1) by adding noise to all 60 frames of input camera poses to improve the robustness of the training to input data not seen before, and (2) by duplicating a number of sequences in which we replace the $n-1$ first frames by a randomly selected frame n within the 60 frames. The second augmentation is critical to force the gating network to extract behavior information from input reference video rather than inferring style from past camera trajectory imported through prediction network; and it guarantees the robustness of our network in testing, since we always assume camera poses of 60 frames ahead of initial frame are already given with the value of the initial frame.

We trained our gating and prediction network simultaneously under a supervised manner, but without behavior labelling. The loss function is defined using the mean squared error between the predicted output and the ground truth as:

$$\mathscr{L}(\tilde{\mathbf{c}}, \bar{\mathbf{c}}) = \|\tilde{\mathbf{c}}_i - \bar{\mathbf{c}}_i\| + \eta \sum_{j=i+1}^{i+29} \|\tilde{\mathbf{c}}_j - \bar{\mathbf{c}}_j\| \tag{6}$$

where $\bar{\mathbf{c}}$ is normalized along each dimension; $\eta = 0.1$, so that the error from current frame has a greater effect in the loss function. The network is trained using the Adam [Kingma and Ba 2015] adaptive gradient descent algorithm. Training is performed for 100 epochs and takes around 30 hours on a NVIDIA GeForce Titan Xp GPU with batch size of 256. We use the exponential learning rate policy with base learning rate set to 0.001 and decay 0.97 after each epoch.

6 EVALUATION

We first evaluate the accuracy and robustness of our cinematic feature estimator using synthetic and real data. We then describe experiments that help reveal what is taking place in our gating+prediction network, and also validate its capacity to reproduce a sequence of behaviors extracted from a reference sequence.

6.1 Evaluating the Cinematic Feature Estimator

This evaluation only focuses on the quality of the deep-learning component of the feature estimator (other components such as camera framing $\mathbf{p}_A, \mathbf{p}_B$ or main character M are less prone to errors).

Accuracy on synthetic data. . First we perform a generality test by measuring the accuracy of the trained network with automatically generated synthetic data. This test data is composed of 500, 000 synthetic data samples. Variation is performed on character animations (a combination of 20 complex animation sequences sampled in time), distances between characters, camera angles and camera composition. Ground truth is computed by projecting 3D skeleton joints in screen space, and converting the traditional 6D camera pose to the corresponding Toric space pose. Quantitative evaluations are illustrated in terms of absolute errors for all the parameters but d_{AB} which uses relative error (see Fig. 7). An observation of minor errors *w.r.t.* the ground truth data shows a decent estimation capacity of the proposed network. The maximum error on θ is around 8°. Since the pitch angle φ varies in a moderate range, its maximum error is only around 2.3°.

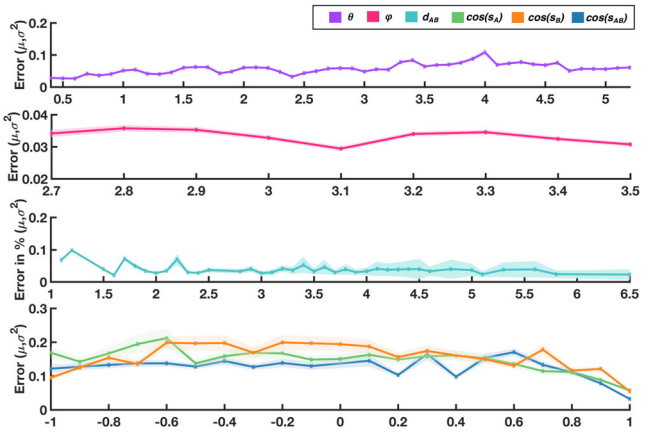

Fig. 7. Generality test on the trained Cinematic Feature Estimator by reporting average and variance in errors between estimation and ground truth, over the range of the variables (θ and φ in radians). Variance is significantly more important on φ as the skeleton 2D features are more difficult to relate to the camera pitch angle.

Accuracy on cinematographic data. To better illustrate our algorithm's robustness, we extract a sequence of camera parameters from *Catch Me If You Can* movie using our Cinematic Feature Estimator. The extracted camera parameter sequence is then retargeted back onto a character animation that has been intentionally designed to replicate the motion of the actors in the corresponding real-world clip. We encourage the readers to watch the side by side comparison between the retargeted animation and original clip in the accompanying video. Though the synthetic nature of the characters may introduce a bias, the results as reported in the video, indicate acceptable errors.

6.2 Evaluating the Gating+Prediction Network

In the context of designing an example-driven camera motion controller, the gating paradigm plays a critical role in specializing the prediction network's behavior with regard to the selected reference clip. To assess this claim, we first test our system without the gating network. Conceptually, our approach is similar to training separate networks for each cinematographic behavior. In order to make a fair comparison, we reshape the prediction network into a shared-bottom structure [Caruana 1997] (see Fig. 8) which has been proven effective and widely used in multi-task applications such as [Dai et al. 2016; Girshick 2015].

The network input, output and loss function are identical to our Gating+Prediction network. The input first goes through a shared feature extractor which consists of a 1380×1024 fully connected (FC) layer. To learn different camera behaviors, we choose individual prediction blocks with two fully connected layers (1024×512 and 512×150 respectively). The ELU (Exponential Linear Units) activation function is used here. We train the share-bottom network using only synthetic data, since the style labelling is a prerequiste in the shared-bottom paradigm. The batch size is set to 256 and Adam optimizer [Kingma and Ba 2015] is utilized. We use the exponential learning rate policy with base learning rate set to 0.001 and decay

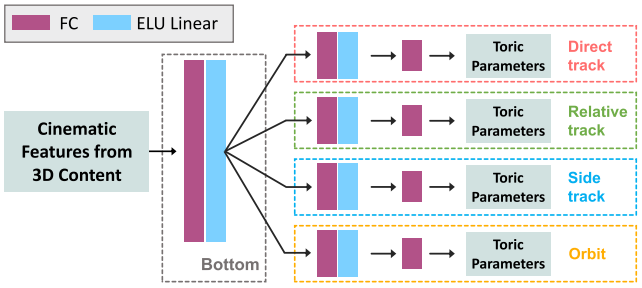

Fig. 8. The shared-bottom adopts a shared feature extractor and independent refiners for different tasks (*i.e.* towers in some cases). The inputs and outputs are identical to our prediction network in Fig 6. To learn different camera behaviors (multi-task learning), we adopt individual refiners (towers) for each behavior.

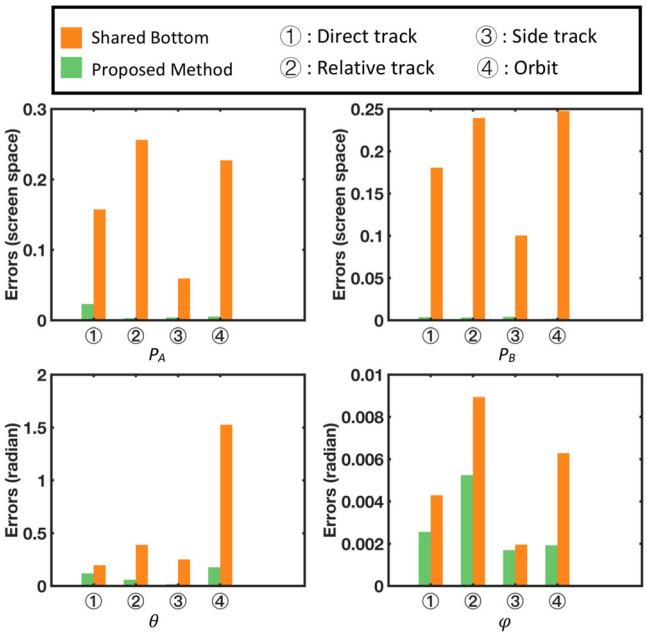

Fig. 9. Statistics shows that our method outperforms shared bottom for all different types of behaviors, especially for side track and orbit.

0.97 after each epoch. 100 epochs are performed in 16 hours in total (10 minutes per epoch) on a NVIDIA GeForce Titan Xp GPU.

A quantitative comparison between Gating+Prediction network and shared-bottom is conducted using the same test dataset. Fig. 9 shows side by side comparison on the distribution of Toric parameters errors. The proposed method outperforms the shared-bottom framework, especially with the side track and orbit behaviors. The reason is the shared-bottom framework tends to learn a commonality, a low-frequency representation of the data and often fails to fit different samples in each task respectively. Typically side track can be further divided according to their direction and initial position, and orbit behaviors could be sub-classified by comparing the frequency and phase.

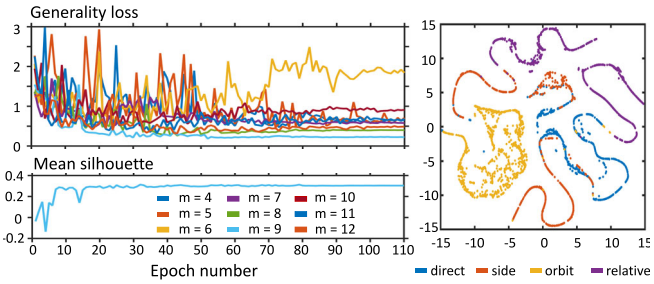

Fig. 10. Number of experts. Top left: our generality loss as a function of the number of epochs for different MoE configuration. Bottom left: mean silhouette coefficient of our test set for the gating output (latent space) when $m = 9$. It may be seen that the network learns to cluster the data even though this isn't explicitly required. Right: the gating output of our test set visualized using t-SNE when $m = 9$.

Next, we examined the prediction network output for different numbers of experts ($m = 4, 5, 6, 7, 8, 9, 10, 11, 12$). Top left figure of Fig. 10 shows statistics of generalization errors for each MoE configuration, and $m = 9$ gains best performance. An interesting outcome of a properly trained Gating+Prediction network is that the network implicitly learns to cluster the input reference clips, despite the fact that it imposes no explicit requirement for separation in the latent space. To measure this clustering ability, we ran each of our test synthetic camera behavior clip through the gating network to obtain its position in the latent space and displayed the results using the mean silhouette coefficient [Kaufman and Rousseeuw 1990], given by $(b - a)/\max(a, b)$, where a is the mean intra-cluster distance and b the mean nearest-cluster distance for each sample. Mean silhouette coefficient measures the difference between the mean inter-class distance and the mean intra-class distance, with best value equal to 1, worst value equal to -1, and values near 0 indicate overlapping clusters.

In more details, after each epoch we calculate the mean silhouette coefficient of the gating network output (latent representation) for our synthetic test set, using the camera behaviors as clustering label. Bottom left figure of Figure 10 plots the mean silhouette coefficient as a function of the number of epochs when $m = 9$. The coefficient is increasing, which indicates that the network implicitly learns to cluster the labeled groups, even though this is not explicitly required. The resulting gating network outputs using 9 experts (after 110 epochs) are presented in right part of Fig. 10 using t-SNE visualisation [van der Maaten and Hinton 2008]. The samples are well clustered in latent space.

6.3 Generality and Failure Case

In order to evaluate the generality of our method, we designed specific use cases by changing the following control variables: (i) Cartesian distance between the characters, (ii) relative size between the characters and (iii) speed of animation. A subset of the testing data is selected, further augmented by uniformly sampling these control variables. We use the generality loss of the network as the metric of our evaluation and display results in Fig. 11.

Character distance. As demonstrated in of Fig. 11(left), the model shows a loss that increases with the distance between the two characters. The loss remains acceptable (0.2 – see Fig. 10 for reference)

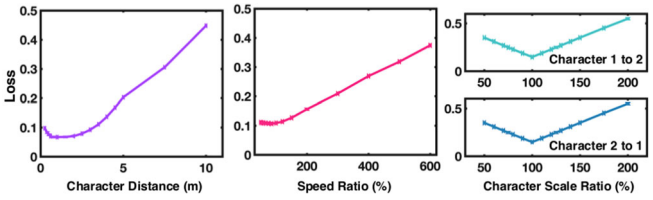

Fig. 11. The evolution of test loss for different values on character distance, animation speed and relative scale parameters. Below an acceptable loss of 0.2, the proposed model enables significant changes on the parameters.

with a distance of 5 meters between characters, covering a large range of scenarios in virtual cinematography. Due to the lack of specific training data, the loss increases with larger distances between characters (> 5m).

Animation speed. We adjust the animation playing speed to mimic different character velocities. As shown in the middle part of Fig. 11, performance decreases when the animation is played at a higher speed (loss is around 0.2 when playing the animation 3 times faster). The proposed method copes well the animation when slowed down and at moderate speeds.

Character relative scales. In most original training data, the two characters are the same height. In applications of virtual cinematography, it is common that characters have different height and shape. To account for this, we observe the loss for different values of relative scales on the characters (*e.g.* character 1 being twice as tall as character 2). Performance in terms of loss is linear with relation to the scale difference between characters and is independent on the character to which the scale is applied: larger scales lead to important loss (see right part in Fig. 11).

Large changes to the selected parameters lead to important loss (> 0.2) and therefore create unexpected camera behaviors (failure cases). Please refer to Section. 8 for an in-depth discussion.

7 RESULTS

In the following we present results for two different representative scenes: a heated dialog scene with distinct spatial configurations between characters (facing, opposite, side by side), and a fast-paced fight scene with multiple characters and a wider range of character poses. Both 3D scenes represent typical setups where complex camera motions can underline the dynamic nature of the sequence. Our camera controller is applied to the two scenes with varying reference inputs. Our results are presented and compared with optimization techniques used in the camera-on-rails approach [Galvane et al. 2015a]. The system runs under Unity 2019.2 and communicates with the trained network (python) using ZeroMQ. At each frame of the animation, the 30 past and 29 future features on the characters, together with the 30 past features on the camera are extracted from Unity and sent to the network which computes the future 30 camera poses. The first predicted camera pose is used to render the current frame, and the process is repeated.

Once the network is trained, and given a vector in the behavior space, the prediction network can compute a new camera pose at a frequency higher than 200Hz. Extracting a behavior from a reference sequence requires a first 0.7*s* overhead to process the first 400 frames but then is insignificant (and can be avoided by storing the outputs of the gating network for all reference clips). Fig. 12 displays two snapshots that illustrate this process.

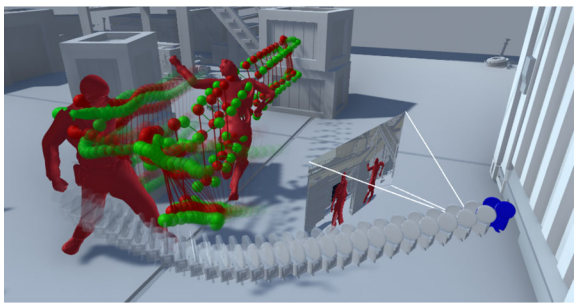

Fig. 12. For each frame of a given animation, we extract the 3D poses of the characters (displayed in red and green) over a sliding window and use our gating+prediction network to determine the next 30 poses of camera (in grey) from which the first one serves as the next camera position (in blue).

Dialog scene. For the dialog scene (4800 frames at 60 fps), we have selected 4 reference sequences which belong to a different style of motion (side track, direct track, relative track, orbit). The cinematic feature estimator was applied on each sequence to extract 2D and 3D features which were then sent to the gating network. A single value of the gating network was then selected in the middle of each sequence, and was used to apply weights on the prediction network. We can use a single value since the reference video has only one style so the latent/style value does not change significantly. Refer to the companion video for a detailed comparison between the 4 behaviors. As illustrated in Fig. 13, in side track mode, the camera keeps looking from one specific side of main character, no matter his/her facing orientation. In Fig. 14 we compared result from direct and relative track with same main character. Relative track in Fig. 14 (top) puts more emphasis on the male character; while direct track in Fig. 14 (bottom) gives the same importance to both characters.

Fighting scene. For the fight scene (4600 frames at 60 fps), the motions of the characters are far more complex and fast-paced. For this scene, we have computed two distinct camera sequences by selecting two different sets of reference videos on which we applied the Feature Estimation and fed to the gating+prediction network. The main difference with the dialog sequence is that the scene is composed of four characters. Switching between characters simply consists in deciding when to switch to a new character and changing the character data sent to the gating+prediction network. The resulting Toric camera parameters will be also applied to the new targets (their positions need to be interpolated to avoid a jump). The switch to one or two new characters only performs well for small transitions (larger changes would require a dedicated approach, see discussion in Section 8). A selection of snapshots is displayed in Fig. 15 with a scene overview at the bottom.

Fig. 13. Side track with different main character. In side track mode, the camera will always look from one side of the main character no matter his/her facing orientation. Green arrow indicates the main character in each sequence.

Fig. 14. Relative vs direct track with same main character. Relative track (top) puts more emphasis on the male character; while the feeling conveyed by direct track (bottom) is more objective. Green arrow indicates the main character in each sequence.

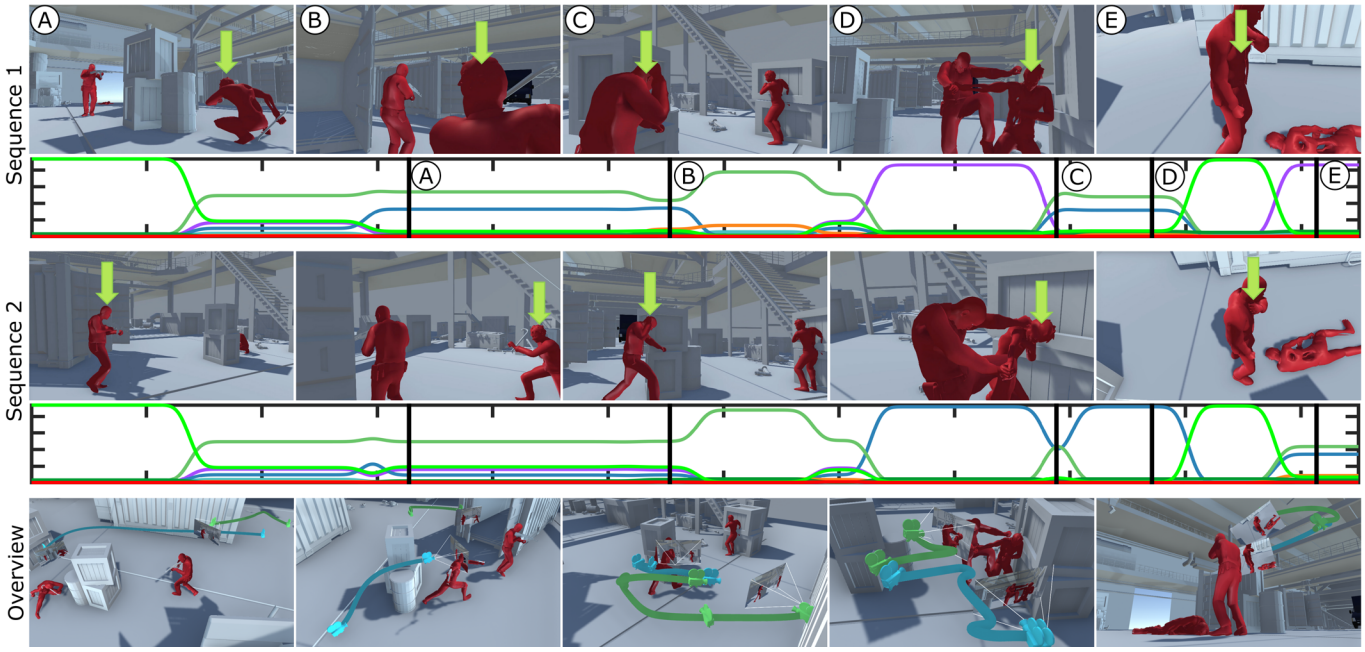

Fig. 15. Snapshots taken from two distinct camera motions computed on the zombie sequence using two distinct sequences of input. The green arrow represents the main character. As viewed in the snapshots, there are changes in the main character throughout the sequence. This enables the designer, through selected clips, to emphasize one character over another at different moments.

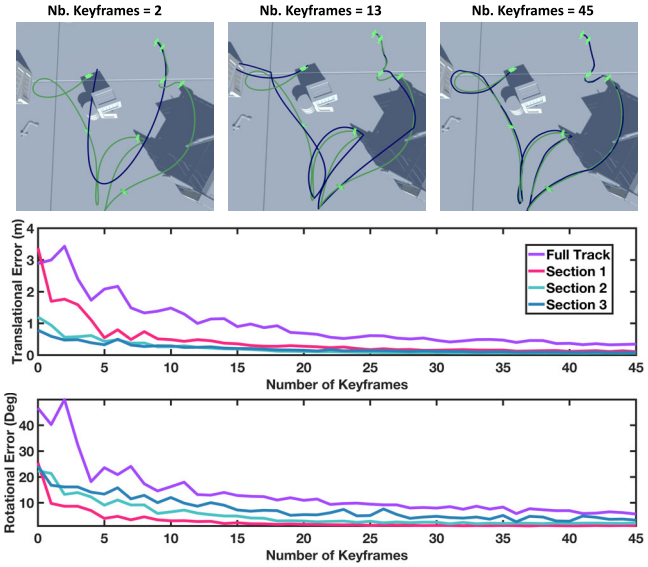

Fig. 16. Comparing average translation and rotation errors between 4 different camera trajectories computed with our deep learning network and the Camera-on-rails optimization-based approach [Galvane et al. 2015a]. The optimization-based approach requires the insertion of at least 15 intermediate keyframes to reproduce the motion on simple trajectories.

7.1 Comparison with Optimization-based Method

Optimization-based methods have often been used in the computation of camera paths when all the scene information is available beforehand [Galvane et al. 2015a; Huang et al. 2016]. To evaluate our approach, we performed a comparison with the camera-on-rails technique [Galvane et al. 2015a] which is the only optimization technique to handle specifically the framing of two targets. Given starting and ending keyframes $(\mathbf{k}_i, \mathbf{k}_f)$ specified by the user, the technique optimizes the parameters of a degree 3 spline curve (the camera rail) by minimizing a metric expressed on the framing along this trajectory. The metric is expressed as a distance between the spline curve and a reference curve which interpolates perfectly the visual properties (distance to targets, angle on targets and image composition) between keyframes $\mathbf{k}_i$ and $\mathbf{k}_f$.

To perform a comparison, we propose to (i) compute a camera trajectory $\mathbf{t}_{\text{ours}}$ with our system by using an example clip (with one or more behaviors), (ii) initialize the camera-on-rails algorithm with the initial and final keyframes of $\mathbf{t}_{\text{ours}}$, and (iii) incrementally insert new keyframes taken from $\mathbf{t}_{\text{ours}}$ until the optimized camera trajectory $\mathbf{t}_{\text{opt}}$ is close enough to $\mathbf{t}_{\text{ours}}$, *i.e.* until the optimized trajectory reproduces the intended camera behavior(s). In essence, this mimics the behavior of an artist wanting to create a trajectory manually by setting keyframes from a reference video and using the camera-on-rail optimizer to create in-between trajectories. We used the code of the camera-on-rails paper in order to chain a number of keyframes, sequentially optimizing every section defined between two keyframes. Original code was only designed to create a path between two keyframes, and does not ensure more than C^0 continuity between optimized sections.

Results are displayed in Fig. 16 where the number of inserted keyframes is reported together with the average distance/angle to our reference trajectory. While simple sequences require only to insert a few keyframes to match the reference path, the Zombie required at least 50 keyframes to reach an average error of $50cm$ in position and 10 degrees in orientation. Results contrast with our approach that only requires to specify a number of reference sequences over time. We refer the reader to the companion video which displays the sequences generated with our and the optimization approach using respectively 5, 65 and 130 inserted keyframes.

7.2 Smoothing Out Camera Paths

Results presented in the companion videos are outputs of our cinematographic camera motion controller. Yet it is important to mention that the camera trajectories have been smoothed out for final rendering, using a sliding averaging window of 20 frames, repeated 3 times, on position and orientation. This removes some artifacts due to the local nature of the Toric space coordinate system, making the camera very sensitive to small jitters in the characters head positions. We refer the reader to the companion video to visualize the difference between the raw output and smoothed output of the example Zombie sequence.

8 LIMITATIONS AND DISCUSSION

Limited analysis of cinematic features. A key limitation of our current approach stems from the reduced number of cinematic features we extract from film sequences. Typically the background motion (*e.g.* optical flow) of a clip is also a relevant cue to identify the type of camera motion, as well as the motion of the characters in relation to the background and the camera. Background motion is also a good indicator of low and high frequency motions which could be learnt. In turn this requires to increase the dimension of the input feature space, *e.g.* using a texture encoding the background motion. On the downside, it also restricts the possibility of applying a behavior learnt in some closed environments (*e.g.* a building), to some outdoor scene since background motions will not match, hence required a much larger training dataset. There are some successful representations of background motions for the specific case of drone cinematography using dense optical flow [Huang et al. 2019b] which could then be used as a regulator on the final motion. In addition, high frequency signals on cinematic features (*e.g.* a vibrating camera) represent a failure case, and cannot be reproduced due to (i) the cinematic feature estimator which outputs a smooth signal and (ii) the incapacity to differentiate the relative motion of the camera or the character without analyzing background motion.

Learning and framing is limited to two characters. Real film footage intrinsically displays more complex scenes than ones with only two characters, hence here we only address a subset of cinematographic possibilities. Yet, we have focused on a non-trivial subset that has not been addressed before and have demonstrated the overall feasibility of the approach. Extending the work to sequences with a sole character only requires to simplify the camera representation, switching from Toric space coordinates to polar coordinates (the Toric space is a generalization of polar coordinates [Lino and Christie 2015]). Gating+prediction could still be trained with both types of sequences

together by using a hot vector representing the number of characters (1 or 2) as how we distinguish the main character. Scenes with more characters would require further investigation, typically by extending the representation of the character features.

Delayed real-time camera motions. Once the network is trained, the execution times are low enough to perform real-time camera animation, *i.e.* having the network to compute a new camera position at each frame. However the network has been designed to predict a position by using as input both the past and future cinematic features on the character animation and the past camera information. This design provides the network with the ability to know in advance the motion of the characters, hence having a camera with anticipation that mimics traits of real film cinematography. The approach therefore requires this 30-frame look-ahead on character animation before computing a camera pose, incompatible with real-time systems yet applicable to systems with a short delay.

Continuous camera behaviors. The proposed approach only focuses on analyzing and generating continuous camera behaviors without cutting between cameras. Encoding cutting behaviors requires additional semantic information which are hard to estimate from real film footage.

9 CONCLUSION

We propose an example-based approach for virtual cinematography that is able to extract camera behaviors from real film sequences and retarget them to virtual environments using a prior learning on a reduced set of camera behaviors. The proposed method has the ability (i) to extract a camera trajectory and cinematic features from film clips with characters using a novel cinematic feature extractor, (ii) to extract camera behaviors composing these film clips by locating them in a latent space of camera behaviors using a gating network and (iii) to animate a virtual camera by applying this sequence of behaviors to a 3D scene, hence transferring elements of camera style of a real film sequence to a virtual film sequence. To the best of our knowledge this is the very first example-based camera motion controller dedicated to 2-character shots.

While our work narrows down the notion of camera behavior to solely a relation between the characters motion and the camera motion, future work will consist in extending the framework to handle a larger set of cinematic features, considering background motion, scene complexity, visual salience, but also other modalities such as speech, diegetic and non diegetic sounds, and ideally consider high representations such as narrative intentions and cognitive aspects.

ACKNOWLEDGMENTS

We 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). We also thank Anthony Mirabile and Ludovic Burg from University Rennes, Inria, CNRS, IRISA and Di Zhang from AICFVE, Beijing Film Academy for their help in animation generation and rendering.

REFERENCES

Daniel Arijon. 1991. *Grammar of the film language.* Silman-James Press.

Jackie Assa, Lior Wolf, and Daniel Cohen-Or. 2010. The virtual director: a correlation-based online viewing of human motion. *Computer Graphics Forum* 29, 2 (2010).

William Bares, Scott McDermott, Christina Boudreaux, and Somying Thainimit. 2000. Virtual 3D Camera Composition from Frame Constraints. In *Proc. ACM Int'l Conf. Multimedia.*

Rogerio Bonatti, Wenshan Wang, Cherie Ho, Aayush Ahuja, Mirko Gschwindt, Efe Camci, Erdal Kayacan, Sanjiban Choudhury, and Sebastian Scherer. 2019. Autonomous aerial cinematography in unstructured environments with learned artistic decision-making. *Journal of Field Robotics* (2019).

Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997).

Marc Christie, Patrick Olivier, and Jean-Marie Normand. 2008. Camera control in computer graphics. *Computer Graphics Forum* 27, 8 (2008).

Jifeng Dai, Kaiming He, and Jian Sun. 2016. Instance-aware semantic segmentation via multi-task network cascades. In *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition.*

Steven M. Drucker, Tinsley A. Galyean, and David Zeltzer. 1992. Cinema: A system for procedural camera movements. In *Proc. Symp. on Interactive 3D Graphics.*

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 *Proc. ACM SIGGRAPH Conf. Motion in Games.*

Quentin Galvane, Rémi Ronfard, Marc Christie, and Nicolas Szilas. 2014. Narrative-driven camera control for cinematic replay of computer games. In *Proc. ACM SIGGRAPH Conf. Motion in Games.*

Quentin Galvane, Rémi Ronfard, Christophe Lino, and Marc Christie. 2015b. Continuity editing for 3D animation. In *AAAI Conf. Artificial Intelligence.*

Ross Girshick. 2015. Fast r-cnn. In *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition.*

Mirko Gschwindt, Efe Camci, Rogerio Bonatti, Wenshan Wang, Erdal Kayacan, and Sebastian A. Scherer. 2019. Can a Robot Become a Movie Director? Learning Artistic Principles for Aerial Cinematography. *Proc. IEEE Int'l Conf. Intelligent Robots and Systems.*

Nicolas Halper, Ralf Helbing, and Thomas Strothotte. 2001. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum* 20, 3 (2001).

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 *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition.*

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 *Proc. IEEE Int'l Conf. Robotics and Automation.*

Hui Huang, Dani Lischinski, Zhuming Hao, Minglun Gong, Marc Christie, and Daniel Cohen-Or. 2016. Trip Synopsis: 60km in 60sec. *Computer Graphics Forum* 35, 7 (2016).

Robert Jacobs, Michael Jordan, Steven Nowlan, and Geoffrey Hinton. 1991. Adaptive mixtures of local experts. *Neural Comput.* 3, 1 (1991).

Leonard Kaufman and Peter J. Rousseeuw. 1990. *Finding Groups in Data An Introduction to Cluster Analysis.* Wiley Interscience.

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

Christian Kurz, Tobias Ritschel, Elmar Eisemann, Thorsten Thormählen, and Hans-Peter Seidel. 2014. Generating Realistic Camera Shake for Virtual Scenes. *JVRB-Journal of Virtual Reality and Broadcasting* 10, 7 (2014).

Christophe Lino and Marc Christie. 2015. Intuitive and efficient camera control with the toric space. *ACM Trans. on Graphics* 34, 4 (2015).

Christophe Lino, Marc Christie, Fabrice Lamarche, Guy Schofield, and Patrick Olivier. 2010. A real-time cinematography system for interactive 3d environments. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Computer Animation.*

Joseph V. Mascelli. 1965. *The five C's of cinematography.* Grafic Publications.

Ian Mason, Sebastian Starke, He Zhang, Hakan Bilen, and Taku Komura. 2018. Few-shot Learning of Homogeneous Human Locomotion Styles. *Computer Graphics Forum* 37, 7 (2018).

Thomas Oskam, Robert W Sumner, Nils Thuerey, and Markus Gross. 2009. Visibility transition planning for dynamic camera control. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Computer Animation.*

Nikolaos Passalis and Anastasios Tefas. 2019. Deep reinforcement learning for controlling frontal person close-up shooting. *Neurocomputing* 335 (2019).

Roberto Ranon and Tommaso Urli. 2014. Improving the efficiency of viewpoint composition. *IEEE Trans. on Visualization and Computer Graphics* 20, 5 (2014).

Craig W Reynolds. 1999. Steering behaviors for autonomous characters. In *Game developers conference.*

Gregory Rogez, Philippe Weinzaepfel, and Cordelia Schmid. 2019. LCR-Net++: Multi-person 2d and 3d pose detection in natural images. *IEEE Trans. on Pattern Analysis and Machine Intelligence* (2019).
Cunka Bassirou Sanokho, Clement Desoche, Billal Merabti, Tsai-Yen Li, and Marc Christie. 2014. Camera motion graphs. In *Proc. ACM SIGGRAPH/Eurographics Symp. on Computer Animation*.
Harrison Jesse Smith, Chen Cao, Michael Neff, and Yingying Wang. 2019. Efficient Neural Networks for Real-time Motion Style Transfer. *Proc. ACM on Computer Graphics and Interactive Techniques* 2, 2 (2019).
Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008).
He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-adaptive Neural Networks for Quadruped Motion Control. *ACM Trans. on Graphics* 37, 4 (2018).

A NETWORK ARCHITECTURE

The full architecture of our network and of the shared bottom network are summarized in the table below, where `Conv1d, BN, MP, FC` denote 1D convolution, Batch Normalization, 1D Max Pooling, Fully-connected layers, respectively. All of the convolution layers use zero padding. k, s are the kernel width and stride. The number of input and output channels are reported in the rightmost column.

Name	Layers	k	s	in/out
Cinematic Feature Estimator	`Conv1d + BN + ReLU + MP`	3	1	28/64
	`Conv1d + BN + ReLU + MP`	3	1	64/128
	`Flatten + FC + ReLU`	–	–	128/128
	`FC`	–	–	128/(1/1/4)
Gating Network	`LSTM`	–	–	5600/512
	`FC + Softmax`	–	–	512/9
Prediction Network	`FC + ELU`	–	–	1380/512
	`FC + ELU`	–	–	512/512
	`FC`	–	–	512/150
Shared Bottom	`FC + ELU`	–	–	1380/1024
Splitted Branch	`FC + ELU`	–	–	1024/512
	`FC`	–	–	512/150

B LCR-NET, POSE ASSOCIATION, FILLING

LCR-Net [Rogez et al. 2019] detects 13 tagged skeleton joints as 2-dimensional on-screen and off-screen positions for each character in a RGB image (see Fig. 3). We utilize LCR-Net to perform frame-by-frame joint estimations on real movie clips. Inaccuracies in estimation occur due to occlusion, characters partially off the screen or lack of image features (out of focus or dark regions). To correct and complete the skeleton, we perform a character association process and a joint filling process.

Pose association and filling. The character association process relies on an L2-norm between normalized joint positions in two images. The first two closest matches are retained as positive assignments. The joint filling process consists in filling character's missing joints in consecutive frames. A naive on image interpolation is applied when the interval frames is small enough (< 3). Film clips for which the data is partially missing or displays strong noise in pose estimations are rejected. A key benefit of LCR-Net is the ability of estimating out of the screen joint positions since LCR-Net optimizes for a full body pose estimation from partial data in the image.

Main character feature estimation. The *main character* feature is computed by selecting the character with the largest average surface on screen over the sequence, for the couple of characters which appear simultaneously over the longest period of time. Surface is computed as $width \times height$ on the axis-aligned bounding box enclosing all joints of the 2D pose estimation.

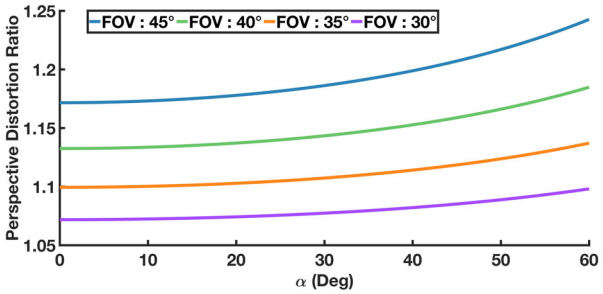

Fig. 17. Perspective distortion ratio w.r.t the FOV (field of view) and framing angle α. In cinematography case, most popular FOVs are ranged from 30° to 45° (roughly equivalent to focal lengths from 60mm to 45mm) and α tends to be smaller when actors are close to each other. These configurations maximally yield 10% and no more than 20% of the error.

C TORIC SPACE

The *Toric Space* is a *manifold* representation of a 5-dimensional camera space dedicated to two targets, and represented as a triplet of Euler angles (α, θ, ϕ). The manifold is determined by a constant angle α between two targets and the camera. The (θ, ϕ) define horizontal and vertical angles around targets, with a camera orientation that ensures screen composition of its two targets. In this paper, α is defined by on-screen positions of targets (p_A, p_B) in Eq. 4.1.

D DIFFERENT CINEMATIC CAMERA BEHAVIORS

Four iconic cinematography behaviors are selected in our paper for synthetic training data generation due to their distinctive correlations between characters and camera motions:

(1) **Direct Track**: camera is mainly in front of characters while keeping the gazing angle towards camera relatively static.
(2) **Relative Track**: similar to direct track only instead of keeping the gazing angle towards camera relatively static, it keeps two gazing angles of characters to be relative static.
(3) **Side Track**: consists of tracking two walking characters at one side perpendicular to walking direction, and simultaneously keeping the framing static.
(4) **Orbit**: means the camera orbits (or semi-orbit) between characters, often happens at long take dialogue scenes.

E DISTORTION OF DIFFERENT FOV

We define the perspective distortion r_d as the ratio of projection length when shifting a vector from the center to the border of image:

$$r_d = (\tan((\text{FOV} + \alpha)/2) - \tan((\text{FOV} - \alpha)/2))/(2\tan(\alpha/2)), \quad (7)$$

where FOV and α are the horizontal field of view angle and framing angle between actors respectively. As shown in Fig. 17, within the range of commonly used FOV configuration, the distortion is mild across different framing angles α.