Automatic Cinematography and Editing in Virtual Environments.

Quentin Galvane

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques-Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Quentin GALVANE

Thèse dirigée par **Rémi RONFARD, Marc CHRISTIE**

préparée au sein **du Laboratoire Jean Kuntzmann (LJK)**
et de **l'École doctorale EDMSTII**

Automatic Cinematography and Editing in Virtual Environments

Thèse soutenue publiquement le **26 Octobre 2015**,
devant le jury composé de :

Karan SINGH
Professor, University of Toronto, Rapporteur
Mateu SBERT
Professor,University of Girona, Rapporteur
Arnav JHALA
Assistant Professor, University of California at Santa Cruz, Examinateur
Philippe GUILLOTEL
Distinguished Scientist, Technicolor, Examinateur
Gérard BAILLY
Directeur de Recherche, CNRS, Président
Rémi RONFARD
Chargé de Recherche, INRIA, Directeur de thèse
Marc CHRISTIE
Associate Professor, University of Rennes 1, Directeur de thèse

RÉSUMÉ

La large diffusion de modèles 3D de qualité ainsi que la mise à disposition de nombreux moyens facilitant la création de contenus animés ont permis de populariser la production d'œuvres cinématographiques 3D. A l'heure actuelle, on peut cependant observer le manque d'outils abordables par tous permettant de traiter la cinématographie (placement des caméras pour l'enregistrement des plans) et d'effectuer le montage de tels contenus (sélection des plans et transitions entre caméras). La création d'un film nécessite la connaissance d'un grand nombre de règles et de conventions. La plupart des outils d'animation n'intégrant pas ces connaissances, le besoin de méthodes automatiques qui pourraient, au moins partiellement, assister l'utilisateur est présent. A travers cette thèse, nous abordons à la fois les problématiques liées à la gestion automatique de la cinématographie et le montage des plans générés.

L'utilisation de caméras afin de retranscrire les actions et événements se déroulant au sein d'un environnement 3D dynamique est une préoccupation importante pour beaucoup d'applications graphiques. Dans le contexte de la simulation de foule, nous présentons une nouvelle approche qui traite du contrôle simultané de plusieurs caméras filmant des groupes d'individus. Nous proposons un système se reposant sur le modèle de "comportements guidés" développé par Reynolds. Ce système permet de contrôler et coordonner localement un ensemble de caméras évoluant dans un environnement dynamique.

Le montage d'un film est une tache particulièrement complexe et méticuleuse qui nécessite une expertise dans le domaine. La formalisation de cette expertise est donc indispensable à l'automatisation du processus. En utilisant la méthode de montage linéaire (ou "continuity editing") comme référence pour l'évaluation du montage, nous présentons une nouvelle approche automatique se reposant sur une hypothèse semi-Markovienne et les principes de programmation dynamique pour déterminer le montage optimal d'une séquence animée.

A partir de notre première contribution, nous proposons une nouvelle approche à la création de *replay* cinématographiques qui utilise à la fois les informations narratives et géométriques extraites des jeux vidéos pour automatiquement calculer des trajectoires de caméra. En combinant ce système avec notre framework de montage, notre solution génère rapidement les *replay* de sessions de jeu.

Enfin, en s'inspirant de pratiques couramment utilisées dans l'industrie du cinéma, nous proposons une nouvelle approche à la planification de mouvements de caméra. Notre solution assure le réalisme des trajectoires en contraignant les caméras sur des rails virtuels. La position et l'orientation de la caméra sont optimisées dans le temps le long du rail pour satisfaire différentes propriétés visuelles. Les plans générés sont ensuite envoyés à notre framework de montage qui produit alors la cinématique.

ABSTRACT

The wide availability of high-resolution 3D models and the facility to create new geometrical and animated content, using low-cost input devices, open to many the possibility of becoming digital 3D storytellers. To date there is however a clear lack of accessible tools to easily create the cinematography (positioning and moving the cameras to create shots) and perform the editing of such stories (selecting appropriate cuts between the shots created by the cameras). Creating a movie requires the knowledge of a significant amount of empirical rules and established conventions. Most 3D animation packages do not encompass this expertise, calling the need for automatic approaches that would, at least partially, support users in their creative process. In this thesis we address both challenges of automating cinematography and editing in virtual environments.

Using cameras to convey events and actions in dynamic environments is a major concern in many CG applications. In the context of crowd simulation, we present a novel approach to address the challenge of controlling multiple cameras tracking groups of targets. In this first contribution we propose a system that relies on Reynolds' model of steering behaviors to control and locally coordinate a collection of autonomous camera agents evolving in the dynamic 3D environments to shot multi-scale events.

Editing a movie is a complex and tedious endeavor that requires a lot of expertise in the field. Therefore, automating the process calls for a formalization of this knowledge. Using continuity editing – the predominant style of editing – as a benchmark for evaluating edits, we introduce a novel optimization-based approach for automatically creating well-edited movies from a 3D animation. We propose an efficient solution through dynamic programming, by relying on a plausible semi-Markov assumption.

Building upon our first contribution we then propose a novel importance-driven approach to cinematic replay that exploits both the narrative and geometric information in games to automatically compute camera paths. Combined with our editing framework, our solution generates coherent cinematic replays of game sessions.

Finally, drawing inspiration from standard practices in the movie industry, we introduce a novel approach to camera path planning. This solution ensures realistic trajectories by constraining camera motion on a virtual rail. The camera position and orientation are optimized in time along the rail to best satisfy visual properties. The computed shots constitute relevant inputs for the editing framework which then generates compelling cinematographic content.

ACKNOWLEDGMENTS

Before getting into the core of thesis, I would like to start by thanking all the people who helped me and supported me throughout these last three years. I first want to express my gratitude to my advisors Rémi Ronfard and Marc Christie for giving me the opportunity to do this thesis and for all the time and energy they spent to guide me through this journey. I express my gratitude towards Marie-Paule Cani, the team leader, who was not only supportive and encouraging but also gave me many opportunities to assist to great international conferences that always boosted my motivation. Thank you also to all the jury members for their precious feedback. Their comments and suggestions helped me to greatly improve this thesis.

A lot of work in this thesis was only made possible thanks to the collaboration with many people. I would like to thank Nicolas Szilas and Nicolas Habonneau for their time and excellent work. I'm also really thankful to the engineers Tristan and Julian for the precious help they provided in times of despair. Obviously, I can not mention my coworkers without acknowledging the amazing work produced by Estelle, Laura, Romain and Adela. Their work was fundamental for my research. Thank you also to Christophe, Vineet and Mike Gleicher, whose advises and suggestions always turned out useful. And a special thanks to the *Christie's*, whose feedback was always most appreciated.

Working at INRIA during the last three years, I came across many wonderful people both in Grenoble and Rennes. I won't go through an extensive list as I'm afraid I might forget to mention a couple and regret it later. But thank you to all the *Coinche*'s players that were there to take a break when I needed one. Thank you to all the runners who accompanied me to the *Bastille*, even when the weather was not the most friendly. And obviously, thank you to my two roommates, coworkers and friends Benjamin and Kévin. I hope it was not too difficult to cope with me and the many sleepless nights I spent working on this very thesis. Overall, thank you to all my friends for all the memorable moments we spent together.

Finally, I would like to thank my family, whose support was an essential component to the success of this thesis. I will never be grateful enough for everything they have done for the past twenty-six years to help me get there. I will now conclude with a huge thanks to Aurélie, my constant source of inspiration and motivation.

PLAN

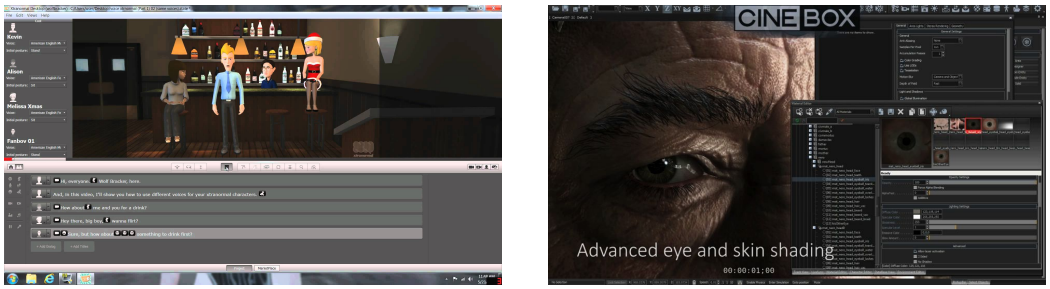
CHAPTER

1

INTRODUCTION

In the last century, the seventh art has seen a spectacular growth, becoming one of the most popular means of storytelling. More recently, the progress in computer graphics, allowing the creation of computer animated movies, has strongly contributed to the increasing expressiveness and quality of this modern type of entertainment. However, whether it is CG or real, movie-making is a long and tedious endeavour that requires tremendous amount of work and expertise in order to pull through the many stages of the production. For the past decades, a fair amount of research has been conducted to ease and accelerate the film production process. In virtual movie-making especially, latest 3D software, game engines, motion capture devices or other virtual production technologies, along with the wide availability of high-resolution 3D models, have revolutionized traditional film-making pipelines. Both industry and public research have eagerly invested in these technologies that not only benefit the movie industry but also enable simple amateurs to become digital 3D storytellers. Gaming companies are also heavily involved in these research topics as the constantly improving quality in graphics urges them to rely more and more on cinematics and cinematographic techniques to enhance the gaming experience and narrative expressiveness of their games.

The rise of the machinima community (portmanteau of *machine cinema*), which relies on the use of real-time game engine as a storytelling tool, has lead to the emergence of low-cost yet powerful alternatives for the creation of virtual cinematic productions. Several projects such as *Xtranormal*, *Moviestorm*, *iClone* or more recently Crytek's *Cinebox*, addressed many of the challenges of virtual movie-making to provide users with high level tools and ease the creation of high quality cinematographic content (see Figure 1.1). To date however, as a large part of the research in the field focuses on modeling, animation and rendering, there remains a clear lack of accessible tools to easily carry out the cinematography (positioning the cameras to create shots) and perform the editing (selecting appropriate cuts between the shots created by the cameras).



(a) *Xtranormal* **(b)** *Cinebox*

Figure 1.1: *Examples of virtual movie-making tools. While some focused on automating as much as possible of the movie-making process (a), others remain highly interactive to provide higher quality results (b).*

These tasks indeed require the knowledge of a significant amount of empirical rules and established conventions, as well as experience with complex computer graphics tools. As such they remain highly interactive and are usually manually performed by graphists and editors. Thus, the cinematographic expertise of the user constitutes a strongly limiting factor for the creation of virtual movies. Providing users with automated tools encompassing this knowledge constitutes a major challenge for virtual storytelling. Not only would it greatly facilitate the creation of cinematographic content in virtual environments but it would also allow users to quickly learn and gain experience in cinematography.

The computation of appropriate cinematography in complex 3D scenes is a key problem in a number of computer graphics applications. The camera constitutes the window through which viewers perceive the virtual environment. Thus, for virtual storytelling, it is the primary tool used to convey the events composing a narrative. Research on camera control has lead to many applications ranging from interactive-based solutions that partially assist users, to fully automatic approaches. In this thesis we investigate the use of fully automatic camera control for storytelling purposes.

Frame composition is the primary concern of cinematographers. As such, the main objective in virtual cinematography is the computation of viewpoints that satisfy layouts of narrative elements on the screen. Due to the dynamic nature of the environments, the other concern in virtual camera control is the motion of the camera. Path planning plays an important role in the shooting of animated scenes. However, even though there is a range of techniques to automatically compute camera paths in virtual environments, to this day, many challenges remain under-addressed. One of them is the use of multiple cameras to convey events in animated scenes in realtime. This objective was partly tackled with solutions either tracking a limited number of targets or restrained to a single viewpoint. None of them however tackled the challenge of coordinating several cameras to optimize the coverage of the scene. Moreover, despite the obvious applications to cinematic replay or other storytelling tools, the use of narrative information to guide cameras was never properly considered. Controlling multiple cameras to convey given communicative goals is a complex endeavor that deserves more consideration. This is the first challenge we address in this thesis.

Finally, very few approaches seriously tackled the problem of generating realistic camera motions. Working with virtual cameras has the advantage of removing physical constraints that usually bound camera movements, hereby giving more space to creativity and expressiveness. However, this convenient property is also a considerable source of errors as the generated path often lacks realism. The second challenge we address in this thesis is the design of new algorithms and techniques that draw inspiration from traditional movie-making practices to automatically generate realistic shots that follow common movie standards.

The last stage of movie making consists in editing the recorded shots to produce the finished motion picture. This task used to be extremely tedious and performed manually by professionals in the movie industry. With the rise of digital media, it has become much faster and accessible to anyone with a computer and proper knowledge of editing principles. Part of the research conducted in the domain now aims at automating the editing process to ease the creation of cinematographic content (see Figure 1.2).

(a) **(b)** **(c)**

Figure 1.2: *Evolution of film-editing. Formerly performed manually by professionals in the movie industry (a), it is currently achieved digitally and also accessible to amateurs (b). Recent research now addresses the automation of the process that would no longer require interactions with expert users (c).*

In virtual cinematography, the problem of cutting and pasting shots from all available cameras has never been addressed extensively. Most 3D animation packages lack editing tools, calling for automatic approaches that would, at least partially, support users in their creative process. The third challenge we address here is the automation of the editing process from virtual cameras using both geometric and narrative information to select the shots that best serve narrative purposes.

The research conducted in this thesis addresses all three challenges through the following contributions, ranging from basic camera control to the automated generation of cinematographic contents.

Steering Behaviors for Autonomous Cameras

In a first contribution, we tackle the challenge of controlling multiple cameras in animated environments. We designed a system based on autonomous cameras which enables the conveyance of events occurring in complex crowded scenes. Our approach relies on Reynolds' model of steering behaviors to control and locally coordinate a collection of camera agents similar to a group of reporters. The key benefit, in addition to the simplicity of the steering rules, holds in the capacity of the system to adapt to the evolving complexity of crowded environment by self-organizing the camera agents to track interesting events.

Continuity Editing for 3D Animation

After focusing on camera control issues, we then address the problem of automatic editing for 3D animation by relying on continuity editing, the predominant style of editing. We review the main causes of editing errors in literature and propose an editing model relying on a minimization of such errors. We make a plausible semi-Markov assumption, resulting in a dynamic programming solution which is computationally efficient. We also show that our method can generate movies with different editing rhythms and validate the results through a user study and a qualitative analysis based on a comparison with a professionally edited sequence.

Narrative-driven Camera Control

This next contribution uses and extends our previous work on camera control and editing to devise a system that generates cinematic replays for dialogue-based 3D video games. The system exploits the narrative and geometric information present in these games and automatically computes camera framings and edits to build a coherent cinematic replay of the gaming session. We propose a novel importance-driven approach to virtual cinematography. Rather than relying on actions performed by characters to drive the cameras (as in most existing approaches), we rely on the importance of characters in the narrative. We demonstrate the features of our system by implementing three camera behaviors and present results obtained by interfacing our system with a full-fledged serious game containing several hours of 3D animated content.

Constrained Camera Paths: Camera-on-Rails

Finally, in our last contribution, we propose an offline camera motion planning system that takes inspiration from live cinematography techniques. Among possible devices, real cinematographers often rely on camera rails to create smooth camera motions which viewers are familiar with. Following this practice, we propose a method for generating virtual camera

rails and computing smooth camera motions on these rails. Our technique analyses character motions and user-defined framing properties to compute rough camera motions which are further refined using constrained-optimization techniques. Comparisons with recent techniques demonstrate the benefits of our approach and opens interesting perspectives in terms of creative support tools for animators and cinematographers.

The outline of the remaining chapters is as follows. After reviewing the related literature and state of the art in chapter 2, we detail our approach of camera control based on autonomous cameras in chapter 3, followed by our work on automatic editing in chapter 4. Chapter 5 details our narrative-driven approach to camera control for cinematic replays. Then in chapter 6, we present our novel approach to camera motion planning using virtual camera rails. Finally, in chapter 7, after discussing limitations and perspectives on future work, we conclude this thesis dissertation.

CHAPTER

2

STATE OF THE ART

Is it for a real movie or a CG movie, movie-making is a complex process that is comprised of many distinct stages including script writing, blocking, staging, shooting and editing. The amount of resources and time required for each of these stages is strongly dependent on the content and the format of the media. The work presented in this thesis exclusively targets virtual environments and deals with two specific phases of the movie-making process: the shooting (cinematography) and the editing. Throughout this state of the art, we introduce the background and review the existing literature related to both of these phases.

This chapter first gives an insight on live cinematography and the common practices in the field. It describes the concepts and grammar used by filmmakers (see section 2.1). After this overview of standard movie-making principles, we focus on the problem of virtual cinematography which consists in shooting in animated virtual environments (see section 2.2). Finally, we conclude this chapter with a thorough analysis of the existing literature on automatic film-editing (see section 2.3).

2.1 CINEMATOGRAPHIC BACKGROUND

Before addressing the problem of *virtual* cinematography and editing, one must be aware of the many conventions, techniques and rules used in real-world movie-making. Compared to others, the seventh art [Can23] is relatively recent and it is only in the second half of the twentieth century that literature on the matter started to define standard practices. Before these conventions were set, movie-makers mostly relied on their experience through trial and error methodologies.

With his book, “The 5 C’s of Cinematography”, [Mas65], Mascelli successfully addressed the issue of "defining, explaining, clarifying and graphically illustrating motion picture filming techniques in an easy-to-understand way”. This first attempt at providing movie-making standards was long considered the main reference for filmmakers. Later, during the past decades, with the popularization of cinema, many books addressed this same issue, focusing on different aspects and trying to formalize common practices [Ari76, Mur86, Kat09, TB93, TB98, Mer10].

When studying filmmaking, it is essential to remember that the primary goal of a movie is to tell a story. There exists many medium through which a story can be told, and in the case of films, cinematography and editing are the tools given to the director to communicate his vision of the story to the audience (2.1.1). After introducing camera placement techniques and shot composition rules (section 2.1.2), we review the fundamental knowledge on camera motion (section 2.1.3) and key editing guidelines (section 2.1.4).

2.1.1 Storytelling and cinematography

Storytelling is the art of narrating a story through one or more mediums (text, speech, images or any other means). As such, movies are considered as a form of storytelling. The main purpose of every filmmaker is to narrate a story ; throughout his book “The Five C’s of cinematography” [Mas65], Mascelli insists on this key principle: “every shot must serve the story”. Cinematography is the tool provided to filmmakers to assist them in the narration of a story. It helps them construct a narrative discourse and convey the story as they want it to be understood.

The theory of G. Genette [Gen72], explains that a narrative is composed of three distinct layers: first, there is the story – also referred to as the *Fabula* – which consists of all events chronologically ordered that occur within the fictional world; then there is the discourse which consists of the subset of events, re-organized in an identical or different temporal order; and finally, there is the narration, which is the act of narrating a story through a specific medium. In [RS14], the authors tackle the issue of expressing a story through different digital media (text, audio, 2D or 3D graphics, etc.). They present several models (that could be applied to any type of media) to transform a *Fabula* (obtained from generated narratives), into a finished mediated discourse. Even though they do not address the specific field of movie-making their general approach relies on cinematographic metaphors. One of their models especially makes use of the movie-making pipeline: a screen-writer transforms a raw story into a refined narrative discourse that describes narrative goals (convey or not a specific action for instance) ; the director is then left with the task of narrating these goals if possible – their last model allows the director to suggest modifications of the narrative discourse in case of failure.

Prior to shooting a movie, directors already know how they want to *narrate* the scenario (or narrative discourse). The way a sequence is being shot plays a significant role in the information and emotion it conveys (see section 2.1.2). Thus, even though the scenario might contain a lot of details on the narrative goals, the director’s vision of the story will always have a major impact on its understanding by the audience. From the same scenario two directors could con-

vey two different stories. Moreover each director has his own style and usually takes decisions beforehand as he often works from storyboards. Directors know precisely how they want to tell the story before even shooting it. Figure 2.1 from the movie *Back to the Future* [ZKSC85], directed by Robert Zemeckis and edited by Arthur Schmidt, shows how precisely Zemeckis shot one of the sequences. Before M.J. Fox, another actor had been chosen to interpret Marty McFly. After five weeks of shooting, the director decided to change the main actor and had to start over. Figures 2.1a and 2.1b show how similar the shots are: the director already knew exactly what he wanted for the end result and shot the sequence only from very specific viewpoints.

(a) *E. Stoltz as Marty McFly* **(b)** *M.J. Fox as Marty McFly*

Figure 2.1: *Shots of Eric Stoltz and Michael J. Fox as Marty McFly in* Back To The Future. *In both cases, the director shot the sequence from the exact same viewpoint to convey a specific idea to the audience.*

Overall, filmmakers' work consists in transforming a raw story into a finished movie that tells the story under their perspective. The following sections aim at introducing the many tools directors have and how they can use them to achieve this goal.

2.1.2 Camera placement and shot composition

A movie is composed of a succession of shots. For each of them, the placement of the camera is essential to the story. It requires finding the best viewpoint to convey actors' actions, reactions and emotions at a precise moment in the narrative, in coherence with previous and following shots. Over the years, filmmakers have defined many stereotypical types of shots that can be defined using shot properties.

Shot properties:

- *Shot size:* it partially describes the framing of a character (or object) by specifying its proportion in the frame. There exists different shot sizes which can be used to convey different emotions to the audience. Figure 2.2 illustrates eight different shot types with a single character. In the "Grammar of the Shots" [TB98], Thompson suggested that all different shot sizes belong to three main categories: Full Shot (including Long Shot), Medium Shot and Close-up. Full shots and long shots allow to see the whole character from head to toe as well as part of the environment. It gives the audience the possibility to follow the character while seeing its surrounding. Medium shots usually focus on the characters' upper body. It allows to both follow the actions the characters are involved in and see their reactions and emotions. Finally, Close-up shots are used by filmmakers to emphasize emotions and trigger the audience attention. In his book [Mas65], Mascelli

dedicates a whole chapter to this type of shot that he considers as "among the most powerful storytelling devices available to the filmmaker".

- *Profile angle:* it is defined as the angle between the camera and the orientation of an actor or a group of actors. As illustrated in Figure 2.3(a), in the case of one actor, it corresponds to the position of the camera around him. The variety of possible shots allows filmmakers to shot scenes from various viewpoints thus conveying different information on the scene and adding some interest to it (a unique frontal shot would probably bored the audience for example). These camera angles can be further extended to the case of two or more actors using the lines of action between them (apex shot or over-the-shoulder shot for instance).

- *Vertical angle:* it can either be low, neutral or high (see Figure 2.3(b)). Low angle shots are taken with a camera usually placed bellow the actor's head. It tends to emphasize their strength and dominance. On the contrary, high angle shots (where the camera shots from above) have the opposite effect; it diminished characters' strength and set them on a weaker position. Taken approximatively from the same height as the characters, neutral angle is used in most cases as it relates the characters to the audience (it sets them at the same level, as equals). This simple change in height of the camera constitutes a powerful tool for filmmakers to emphasize emotions and convey different impressions.

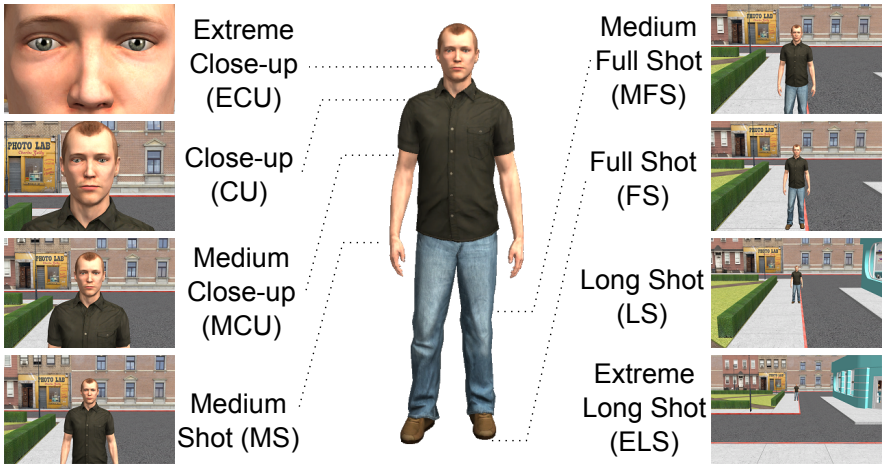


Figure 2.2: *Examples of eight different shot sizes: Extreme Close-up, Close-up, Medium Close-up, Medium shot, Medium Full shot, Full shot, Long shot, Extreme Long shot*

Most of the examples given so far relate to one-actor situations. But all the vocabulary can also be extended to the case of multiple actors. This *grammar of the shot* was formalized by [RGB13] with the Prose Storyboard Language (PSL). The PSL is a formal language used to describe movies shot by shot, where each shot is described with a unique sentence. It covers all possible types of shots and also handles camera movements. Figure 2.4 gives a few examples of PSL sentences used to describe camera shots.

This cinematographic grammar is used to describe what the camera is filming. But the framing of shots also have to obey some rules to ensure the aesthetics of the movie. These

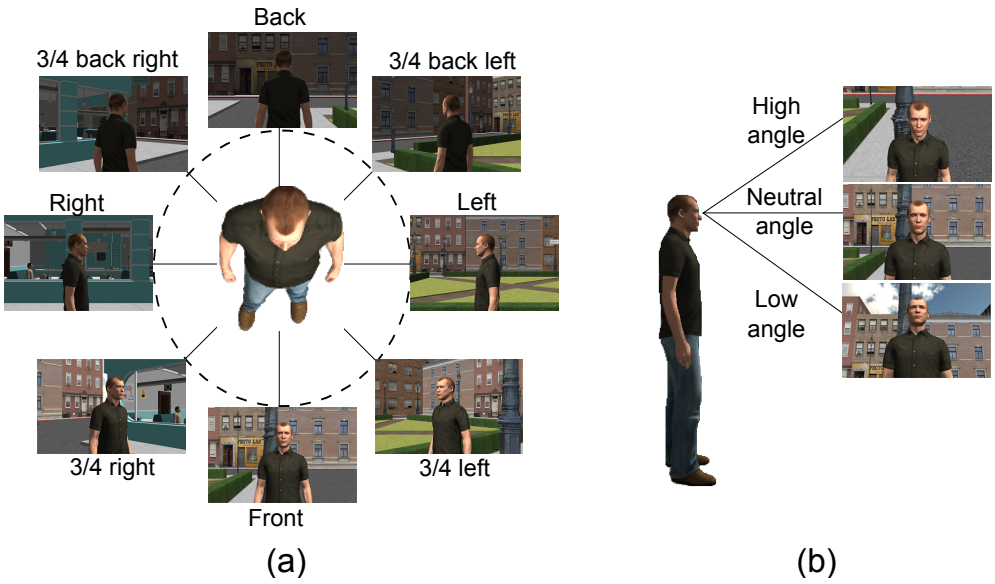
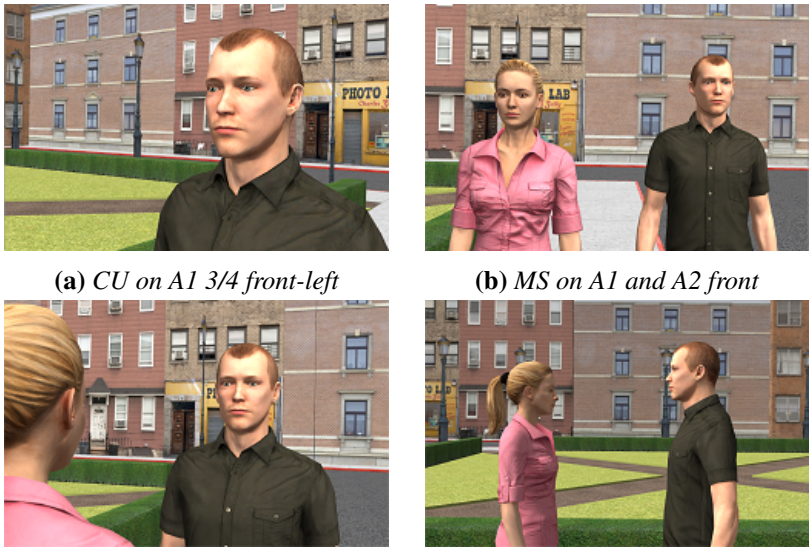


Figure 2.3: *Variety of camera angles: profile angles (a) and vertiacl angles (b)*



(a) *CU on A1 3/4 front-left*

(b) *MS on A1 and A2 front*

(c) *CU on A2 3/4 back-right screen-left and A1 screen-right*

(d) *MS on A1 left screen-right and A2*

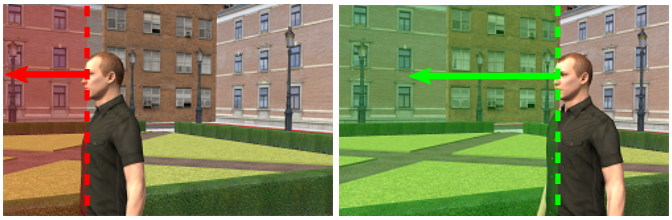
Figure 2.4: *Description of shots using PSL with two actors A1 and A2*

rules aim at ensuring that the shots are properly balanced. The composition of the frame can be broken down into two distinct entities (horizontal and vertical composition) that both obey specific rules:

- *Headroom:* it is the space between the top of the head of a subject and the top of the screen. A large headroom results in unaesthetic framing as a big amount of screen-space is wasted on insignificant elements. On the contrary, no headroom tends to disturb the audience and might result in cropping characters' head.

- *Lookroom:* it is the empty space in the direction the subject is facing or moving. To obtain a proper horizontal composition, the lookroom should remain sufficient. The idea behind this principle is that the gaze of the characters should be accounted for in the balance of the shot.

Another well-known composition rule is the rule of thirds. It states that important compositional elements (characters' eyes for instance) of a frame should be positioned along imaginary vertical and horizontal lines placed at the thirds of the screen. This rule of thirds is complementary to the lookroom and headroom rules as seen in Figures 2.5 and 2.6.



(a) **(b)**

Figure 2.5: *Examples of shots with different lookroom. Insufficient lookroom is observed when a character positioned on the left is looking towards the same side of the frame (a) ; appropriate lookroom is obtained when a character positioned on the right side of the screen is looking toward the opposite side (b)*

(a) **(b)** **(c)**

Figure 2.6: *Examples of shots with different vertical composition. (a) The top of the head of the subject touches the top of the frame and the eyes are not at the third of the screen ; (b) the shot contains a proper headroom and the eyes are nicely aligned at the third of the screen; (c) the headroom is too important and result in a poor shot aesthetic*

This thesis mainly focuses on these few principles and does not account for other rules regarding the lighting, the salience, the background staging, etc. The last criterion that is being used in this work is the *Hitchcock principle*. As seen in section 2.1.1, the goal of a movie is to communicate a story through a specific point of view. To reach their communicative goals, filmmakers determine camera placement by narrative significance. In other words, filmmakers decide where to place a camera depending on how it conveys the story. This idea was

also expressed by film director Alfred Hitchcock. What we refer to as the *Hitchcock principle* [TS67, Haw05, DeL09] states that *the size of a character on the screen should be proportional to its narrative importance in the story*. Selecting a shot then consists in finding the one that best balances the importance of the characters with their perceived size on the screen to reach what we describe as an "Hitchcock equilibrium".

2.1.3 Camera movements

In the previous section, we discussed essential cinematographic principles that, all together, constitute a strong basis on shot composition. Yet, we only looked into the simple case of static composition. Due to the strongly dynamic nature of movies, it is easy to understand the necessity of using dynamic shots. The dynamism can sometimes come from the characters motion within the frame – some directors such as Steve McQueen, Tsai Ming-liang or even Orson Welles are well-known for there extensive use of long static shots – but in many cases, it is induced by camera motion. In [TB98] Thompson defines three categories of shots:

- *Simple shot:* it is taken from a static camera that can neither move nor turn. Any change in the composition necessarily comes from movements of the characters or objects being filmed.
- *Complex shot:* it is taken from a fixed camera with three degrees of freedom: tilt, pan and zoom. Tilt affects the vertical composition of the shot as the camera rotates up and down. Pan is obtained by rotating the camera horizontally and thus affects the horizontal composition. And finally zoom-in and zoom-out narrow or widen the field of view.
- *Composite shot:* it is taken from a moving camera. The number of degrees of freedom depends on the type of support the camera is attached to (see Figure 2.7). With a dolly for instance, the camera is able to move in one direction. It can be used to provide consistency in the framing by tracking characters or on the contrary to transition between two different framings (from a medium shot to a close-up for example). Other more complicated movements can be achieved with the use of camera devices such has steadycam or crane that might require several camera operators.

(a) **(b)** **(c)**

Figure 2.7: *Examples of various camera devices used in the movie industry: (a) camera cranes are used to shot sequences with important changes in viewpoints' height ; (b) steadycam assist operators to film without any shaking artifacts ; (c) camera tracks are used to make precise dolly shots and often requires several camera operators*

As mentioned before, camera placement is the key to communicate a story. The way the director places the camera will affect the way the audience understands the story. For the same reasons, the way the camera moves affects the global understanding of the story. As a result, the use of camera motion should always be motivated. It has to serve a purpose, otherwise, the overuse of camera motion might only confuse the audience.

2.1.4 Editing principles

Film-editing is the task of selecting shots to combine them into sequences that finally create a finished motion picture. As Bordwell and Thompson stated, it is "the coordination of one shot with the next" [BT01]. Originally, before the appearance of digital media, the editing used to be hand-made by physically cutting and gluing raw footage (or "rushes") from different cameras (see Figure 2.8). Nowadays, this task is computer-assisted but the principle remains the same.

Figure 2.8: *Movie editing before the apparition of digital media*

As a standard 90 minutes movie contains in average a thousand edits (*i.e.* cut from one camera to another), it is easy to understand the importance of editing. For the same reason shot composition follow standard practices, film editing obey to its own set of rules and conventions. There exist many different styles of video editing (each with its own standards). The *dragnet* style [Mur86] for instance, adopts a reactive approach where cuts are dictated by new narrative events (such as new actions or new speaker). In this thesis, we focus on *continuity editing*, the predominant style of editing in Hollywood. This less mechanical and more subtle editing style has been extensively studied by [Smi05].

Editing is considered by many as the *invisible art* [O'S09]. Its main purpose is to convey a series of actions and events from different viewpoints while keeping cuts unnoticed by the audience. According to Smith, the goals of continuity editing are to *"minimize the awareness of cuts, create the perception of continuity across a cut and ensure that continuity is not violated as a consequence of a cut"* [Smi05]. To accomplish these goals, continuity editing relies on several rules (later illustrated in chapter 4.4):

- *Jump cuts:* In film editing, a jump cut is defined as a cut in which two sequential shots of the same subject are taken from similar camera positions. This type of edit gives the effect of jumping forwards in time and thus should be avoided.
- *Screen continuity:* Spatial continuity is essential to ease the transition between two shots and enforce the visual fluidity of the cut. It prevents the disorientation of the audience.
- *Motion continuity:* Motion of the characters on the screen also affects the quality of a cut. Transitions between two shots where one or more characters have a different perceived motion (in screen space) have a disorienting effect on the audience.

- *Gaze continuity:* Visual discomfort may also happen in a transition between two shots when characters involved in both shots are looking in different directions on the screen before and after the cut.

- *Left-to-right ordering:* The left-to-right ordering of characters is another important factor to enforce visual continuity. Characters whose relative screen positions are reversed after a cut appear to be jumping around, which attracts attention to the cut [Smi05] – this criteria is also known as the 180-degree rule which states that a camera should never cross the line of interest [Ari76].

In addition to the rules and editing principles previously mentioned, the last century of cinema has seen the emergence of patterns in the way films are shot and edited. These stereotypical ways of filming specific actions are call idioms. They are predefined sequences of shots used to convey series of actions and events. Figure 2.9 gives a simple example of a two-character dialogue idiom. It switches from an apex shot to successive over-the-shoulder shots and internal shots in order to always keep the character speaking in the frame and intensify the dialogue. There also exist idioms to film fighting scenes, 3-characters dialogue, characters walking and any other type of event that usually occur in movies.

(a) **(b)** **(c)** **(d)** **(e)**

Figure 2.9: *Example of a dialog idiom. After an apex shot (a), the editor uses successively over-the-shoulder shots (b and c) and internal shots (d and e) to intensify the dialog.*

Finally, the desired pacing of a scene has a direct influence on the editing. It sets the rhythm (or tempo) of the scene and has an important influence on the emotional response from the audience. For instance, an action scene with a lot of movement involved would probably be better conveyed with a higher pace to emphasize the idea of movement. On the opposite a dramatic scene would better communicate the tension through longer shots. Film scholars have extensively studied shot durations in cinema and found it to be an important parameter of film style. They found that the distribution of shot durations in a movie sequence approximately follows a log-normal distribution [CDN10, Sal09], whose two parameters can be used as a stylistic signature for movie directors.

2.2 Controlling virtual cameras

The challenge of controlling virtual cameras has triggered the interest of researchers in the past decades which results in a wide literature in the field. A significant part of this literature focuses on the problem of interactive control where the purpose is to assist a user in the placement of a camera. This issue was tackled under three different angles. The most common approach is the direct control of the camera parameters through various metaphors (*eyeball in hand*, *world in hand*, *flying vehicle*, etc.). A large spectrum of these solutions was reviewed in [WO90]. Other more sophisticated solutions adopt the "Through-The-Lens" paradigm introduced by Gleicher and Witkin in [GW92]. It consists in manipulating the 3D camera by controlling and constraining features in the 2D image seen through its lens. Finally, a more recent approach consists in

assisting camera control through physical controllers similar to real-world cameras [LCRB11b] (see Figure2.10).

(a) **(b)**

Figure 2.10: *Virtual camera system used to interactively explore virtual environments. It was made famous through its extensive use by movie director James Cameron during the production of* Avatar *(a). It is also used for previsualization and fast prototyping purposes [LCRB11b] (b).*

In the context of this thesis we are not interested in low-level interactivity with a potential user. The goal of our work is to fully automate camera placement and motion. This thesis will not focus on interactive camera control. The first part of this section is dedicated to static camera placement (section 2.2.1). We review constraint-based, optimization-based and algebraic solutions to the problem of positioning the camera at a precise moment in time. We then consider camera motion and look at the work conducted in real-time camera planning (section 2.2.2). Finally we review the related work that deal with camera planning as an offline process (section 2.2.3).

2.2.1 Automated camera placement

Before looking at ways to control camera motion it is mandatory to look at the problem of static camera placement. Positioning a camera in a 3D environment to satisfy a precise 2D shot composition is a challenging issue that requires solving a 7D problem as camera set-ups have 7 degrees of freedom (*dof*): 3D position, 3D orientation, 1D focal.

The early work of Blinn [Bli88] presents the first mathematical formalization of the problem and addresses it using vector algebra. In the paper, he addresses the specific challenge of placing a camera in a virtual environment to film two entities with constraints on their onscreen composition, on the distance to one of the entities and on the onscreen orientation of the other. He proposed an iterative algorithm that computes an approximate solution to this onscreen composition problem. This algorithm however might compute undesired results for some singular cases.

More recently, Lino and Christie proposed a toric surface model that efficiently generates a range of viewpoints corresponding to the exact on-screen composition of two or three targets [LC12]. Unlike Blinn's algorithm, their solution is not restricted to one particular case and can be used with different constraints on the onscreen layout of the targets, such as their orientation or their scale (*i.e.* their distance to the camera). To demonstrate their solution, they first consider the problem in a 2D world. The screen of the camera is then a 1D segment. Using the *inscribed angle theorem*, we know that all camera positions that are solutions to the system define an arc-circle with the targets at its extremities as explained by [Fis] and illustrated in Figure 2.11a. Using the same reasoning, the authors extended this solution to the 3D problem by rotating the arc-circle around the axis defined by the two targets. For a given focal length

and aspect ratio, the manifold surface illustrated in Figure 2.11b contains all the solution to the on-screen composition problem. Other constraints can be satisfied by searching this manifold space defined by two angles θ and ϕ, reducing the space search from 6D to 2D.

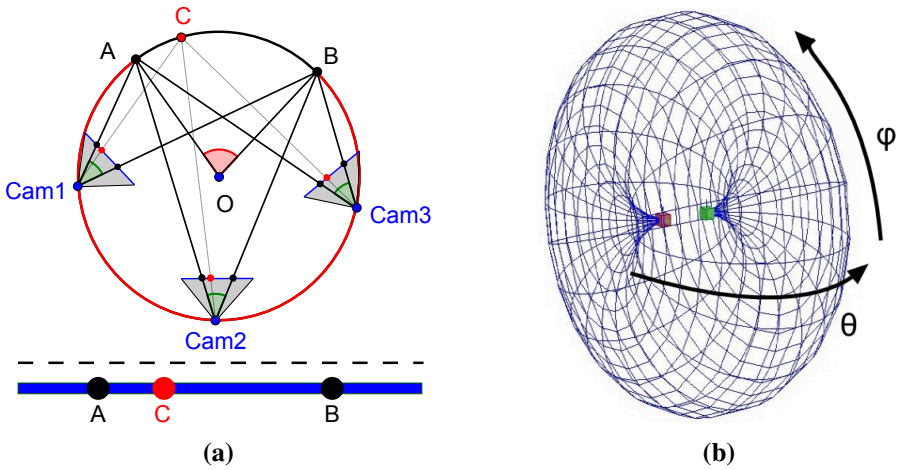


Figure 2.11: *Range of solutions for the exact composition of two subjects on a screen in a 2D and in a 3D world [LC12]. (a) In a 2D environment, all the solutions are along the arc-crircle AB, centered in O ; (b) in 3D, solutions are on a toric manifold surface*

Algebraic approaches offer fast and efficient solutions to automatic viewpoint computation. Nevertheless these methods are generally limited to a small number of targets and do not handle cinematographic issues such as obstruction or occlusion. To tackle these challenges, researchers have devised many systems relying either on optimization or constraint-based approaches. In a state of the art paper, Christie and Olivier [CO09] reviewed and classified the literature. Along with a thorough analysis of the previous work, they presented the classification illustrated in Figure 2.12.

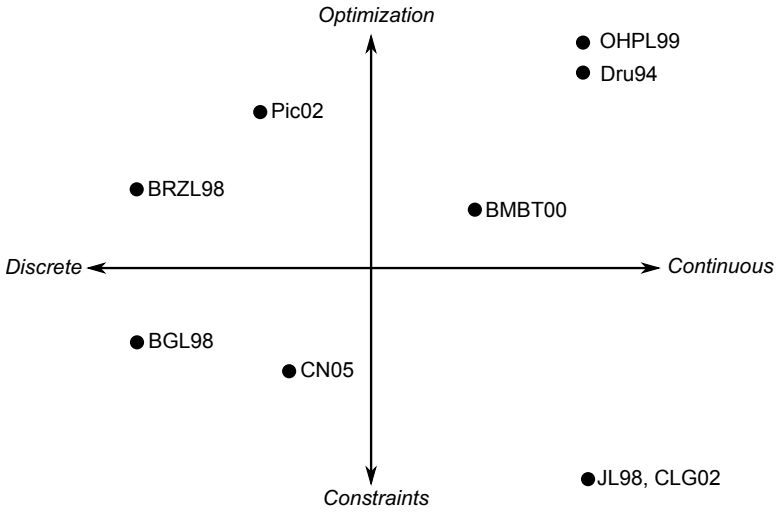


Figure 2.12: *Classification of viewpoints computation systems approaches by [CO09], considering two axes: from discrete to continuous techniques and from constraint-based to optimization techniques.*

Optimization-based techniques are considered soft solving techniques, as the best solution is computed using heuristics that measure violations of each property (*i.e.* an approximate solution can be returned when there are no exact solution). The optimal solution is computed by maximizing a set of objective functions derived from visual properties. With CAMDROID, Drucker *et al.* [DZ94, DZ95, Dru95] proposed a system relying on constrained optimization techniques. The authors encoded visual properties through constraints and objective functions to numerically solve the problem using Newton's method. Later, introducing their CAMPLAN system [OHPL99] (improved in [HO00]), Olivier *et al.* proposed a purely optimization-based solution. Given a set of visual properties (position, size, orientation or visibility of objects on the screen), the system uses a genetic algorithm to optimize the camera position by encoding its parameters as a gene. Similar to these approaches, Bares *et al.* [BMBT00] used storyboard frames to define objectives on shot composition. The search space is recursively sampled to search for optimal solutions using the objective functions to evaluate visual properties. Pickering *et al.* [Pic02, PO03] proposed a discretization approach that only explores a fraction of the search space: after pruning inconsistent areas (see Figure 2.13), the system performs a genetic search to find the optimal solution. More recently, [BDER08] addressed the issue using a Particle Swarm Optimization (PSO) algorithm which proved more efficient than previous solutions.

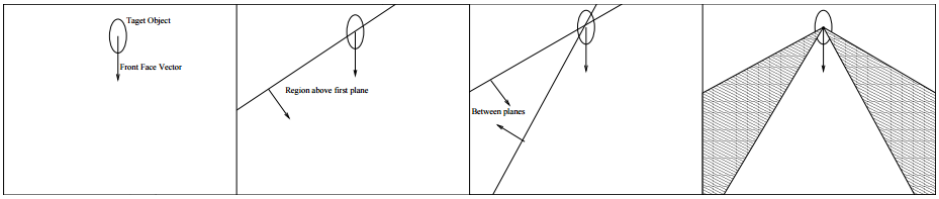


Figure 2.13: *Construction of a 3/4 front shot using [Pic02] prunning approach. The search space is progressively reduced to the grey area*

At the other end of the spectrum, constraint satisfaction techniques are considered as hard solving techniques. They perform an exhaustive exploration of the search space and either return the set of solutions to the problem or the guarantee that the problem has no solution. Nevertheless, hierarchical constraint approaches are able to relax some of the constraints and find an approximate solution. For instance, with CONSTRAINTCAM, Bares et al. [BGL98] proposed a partial constraint satisfaction system that can relax weak constraints to return alternative solutions when all constraints cannot be satisfied. When no solution is found, the system gathers pairs of conflicting constraints and relaxes the weak ones to compute an approximate solution (this constraint relaxing process is iteratively performed until there remains no conflicts). In [CN05], Christie and Normand introduce the notion of *semantic volume* that is defined as *"a volume of possible camera locations that give rise to qualitatively equivalent shots with respect to to cinematographic properties, i.e. semantically equivalent shots"*. This approach consists in defining the search space using cinematographic terms. Semantic volumes are created to satisfy visual properties (such as occlusion, framing, orientation of objects or shot sizes) specified by the user (see Figure 2.14). The solution that satisfies all properties is given by the intersection of the semantic volumes associated with these properties.

All these general solutions rely on different metrics to evaluate the quality of shots regarding various visual properties. Thus, the quality and computational efficiency of the proposed approaches are strongly dependent on the quality of the metrics being used which often turns out to be a limitation factor. To address this issue, Ranon et al. [RCU10] recently focused

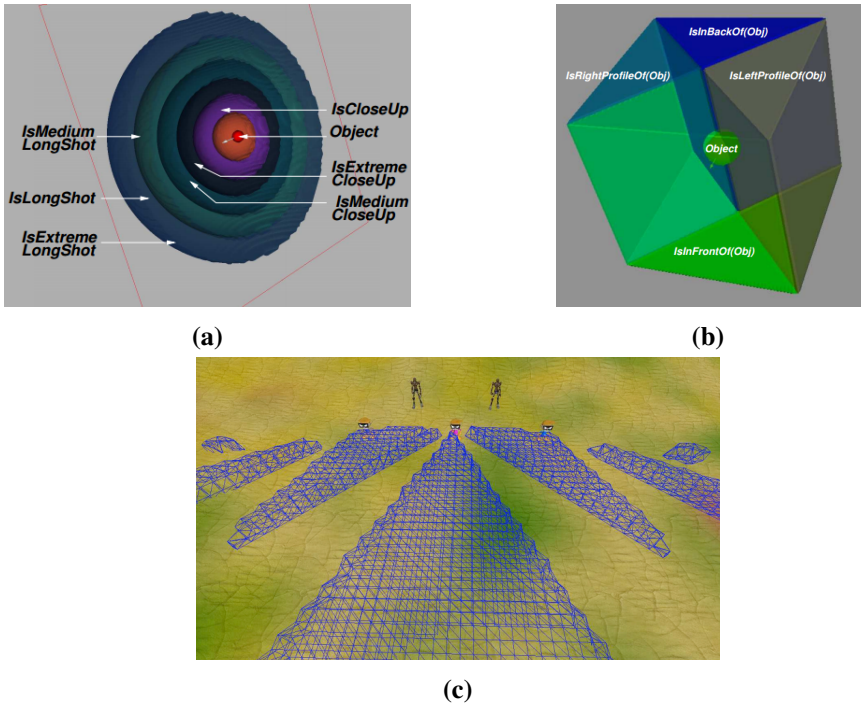


Figure 2.14: *Semantic partitioning of the space search [CN05]: (a) encoding of shot sizes ; (b) encoding of relative camera angles ;(c) camera volumes yielding no occlusion between the characters on the screen.*

on the computation of relevant heuristics and proposed a new rendering-based technique to accurately measure the satisfaction of common visual properties.

2.2.2 Realtime camera planning

The computer graphics research community has been showing an increasing interest for techniques to automatically move virtual cameras in 3D environments in realtime. A first and fundamental requirement of any camera control system is the ability to track (*i.e.* maintain visibility) one or several targets. Techniques vary according to the knowledge available to the camera control system (known *vs.* unknown environments, known *vs.* unknown trajectories of targets) and to the static or dynamic nature of the 3D scene. The complexity and unpredictable nature of interactive 3D animated scenes, coupled with the necessity to convey contents matching some viewpoint quality metrics, has been addressed through different reactive approaches where camera set-ups are recomputed each frame to respond to external stimuli from their environment. As they share the same constraints and requirements, results in robotics have greatly inspired such techniques. For Instance, in [CM01] Courty and Marchand propose a solution based on visual servoing – also known as vision-based robot control [ECR92]. Their solution integrates constraints in the camera trajectory to address non-trivial computer animation problem.

Another type of robotic-like approach was also applied to the camera planning problem: the Probabilistic Roadmap Method (PRM) [LT00, SGLM03, NO04, LC08]. The PRM is a motion planning algorithm in robotic that solves the problem of computing a path between the starting configuration of a robot and a goal configuration while avoiding collisions with

obstacles. It randomly samples the space and tests whether the nodes (samples) are in free space. It then connects each node with its nearby neighbours to create a graph. A graph search algorithm is finally applied to find the best path between the starting configuration and the goal. Figure 2.15 illustrates how PRM works and how it can be applied to camera planning. In [LT00], Li *et al.* used PRM to compute collision-free paths in order to correct a user's input. The method however only works with static obstacles. Li later addressed the issue in [LC08] where they compute a local probabilistic roadmap that is defined in the basis of the target (as the target moves, the whole roadmap moves). Camera planning is then performed locally in this roadmap and globally checked against visibility issues. Another inconvenient of the PRM is that it generates unaesthetics path. Since the graph is built on randomly sampled nodes linked together, the computed path is only made of consecutive segments. This issue was tackled in [NO04] by Nieuwenhuisen and Overmars who use circular blending at the nodes of the graph to smoothen the camera path. Geraerts also addressed this issue in [Ger09] by using another roadmap planner: the Corridor Map Method (CMM).

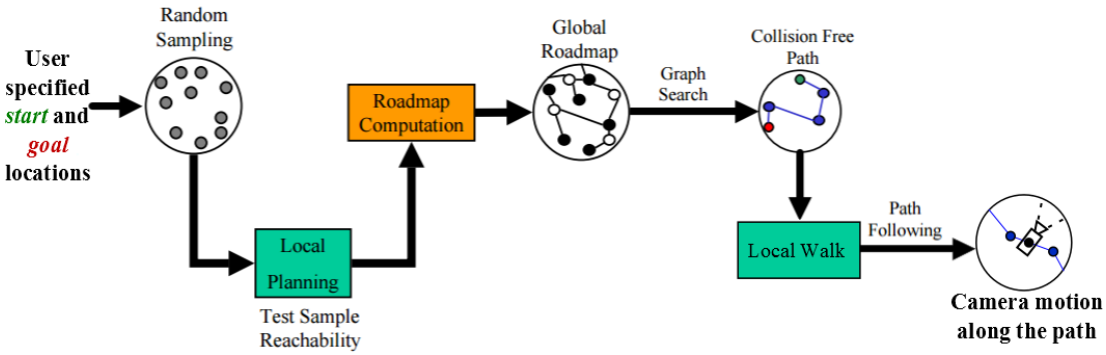


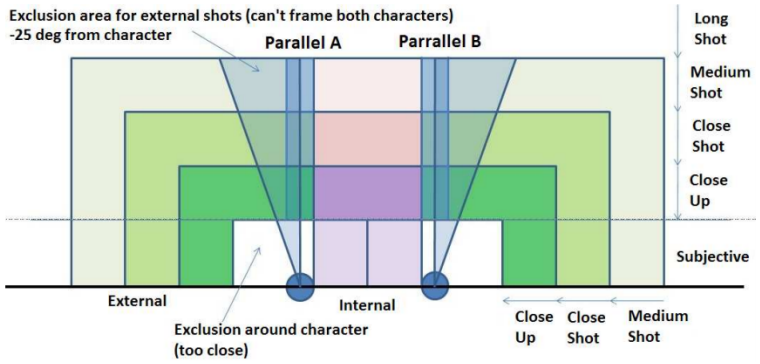
Figure 2.15: *Camera path are computed with PRM by first sampling the space between the initial and goal locations [SGLM03]. The roadmap is then created by linking together close nodes with no obstacles in-between. After a graph search is perform, the camera can follow the computed path.*

Still relying on robot path planning literature, researchers have looked at another approach: Artificial Potential Fields (APF). APF is a reactive approach where the robot's path is not planned but is instead the result of its interactions with the environment. It mainly consists of force vector fields caused by obstacles or target positions. In [BJ09], Burelli and Jhala exploited APF by using force fields to express frame constraints and control both the camera motion and the visual composition. Repulsive force fields are used to push the camera away from obstacles whereas attractive force fields are used to push towards a target position. Frame constraints are expressed using forces affecting both the camera position and orientation in order to reach the desired frame composition.

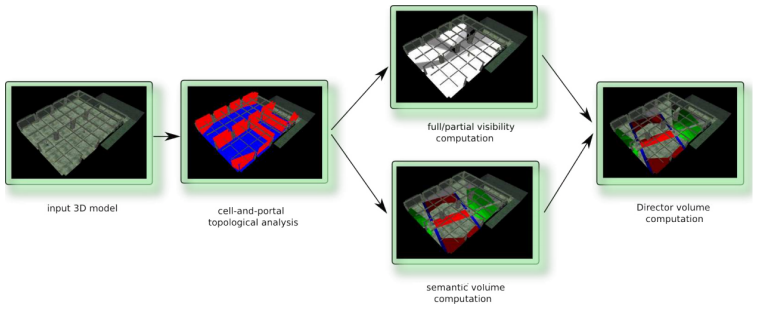
In 1997, [LGBBL97] addressed the problem of computing robot motion strategies and proposed a solution to maintain the visibility over a moving target. It relies on the prediction of future target position and move in the environment. Similarly, [BGBLT97] presents the *Intelligent Observer* (IO), a mobile robot which moves through an environment while autonomously observing moving targets selected by a human operator. The observer moves the camera so that it would see most of the future target positions. Restricted to the task of tracking a single target, [VMGL12] recently extended these methods to the case of a potentially large group of targets in virtual environments. The authors proposed the first group following method that

partitions the workspace into a set of *Monotonic Tracking Regions* (MTR). MTR are regions in which every point can be collectively visible from a trajectory in the MTR. The group following problem is then cast to a small linear-programming problem in each MTR.

Lino *et al.* [LCL*10] presented a solution to automatically position a camera in a 3D environment for given specifications on visual properties (on-screen layout of subjects, vantage angles, visibility, scale) in an animated scene. They introduced the *Director Volumes*, built upon [CN05] semantic volumes which provide a characterization of the space of viewpoints through cinematographic terms (see Figure 2.16a), and combined with full or partial visibility computation (see Figure 2.16b). Continuous transitions between Director Volumes is performed by building a roadmap from a topological analysis of the environment and the partitioning of the space (obtained from subdivisions of the semantic volumes).



(a) *Design of semantic volumes for two key subjects represented as circles*



(b) *Steps leading to the creation of Director Volumes from an input 3D model*

Figure 2.16: *Examples of Semantic and Director Volumes proposed by [LCL*10] for a classical interaction between two key subjects.*

Other real-time systems have addressed the problem of controlling cameras in a virtual animated environment through various methods. In [LZ14] for example, Lixandru *et al.* proposed a physics based model (physical camera rig) to simulate a reactive camera that is capable of both tracking a moving target and producing plausible response interactively to a variety of game scenarios. With a completely different approach, [HHS01] used a reasoning process on visual properties coupled with hardware projections to track a target in a reactive way. [OGK*10] proposed a solution based on a viewpoint entropy map to evaluate the goodness of viewpoints and find the goal position of the camera. It then uses the A* algorithm to find the best path on a roadmap graph. Despite providing interesting novel approaches, these solutions remain limited to the case of a single target. The tracking of multiple targets has been tackled in [CNO12] by performing a sampling process in the space of camera viewpoints using shadow

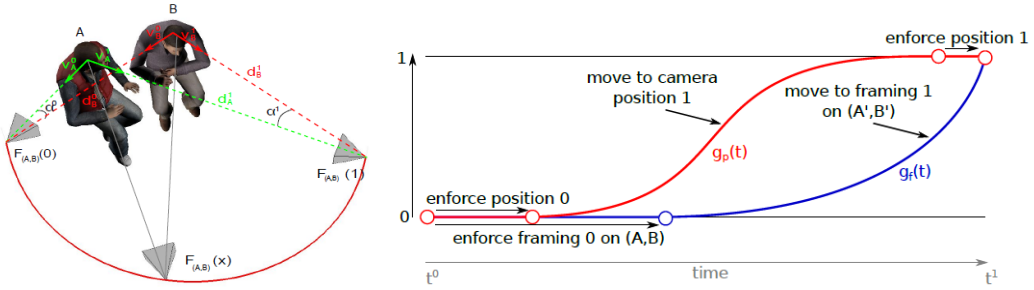
maps principles to compute the visibility of targets. The method provides an efficient evaluation of the visibility within a restricted search space (a set of close candidate camera locations) and composes the resulting visibility information to find the best camera position. Bares and Lester have also worked on several systems relying on real-time camera control. In [BZRL98], Bares *et al.* developed a cinematic task modeling framework for automated real-time task-sensitive camera control. Their camera planning system is composed of three elements: a shot composer that interprets the shot composition guides to place the camera, an occlusion checker that forces cuts to other cameras when occlusions are detected and a transition planner that creates motion splines to smoothly transition between viewpoints. And in [BGL98], they presented CONSTRAINTCAM that uses a partial constraint satisfaction system which can relax weak constraints to ensure a solution can be found. The user study they conducted however highlighted that the continuity and transitions should be improved.

In this section we did not review standard methods used in video games as they address a completely different challenge. They aim at controlling the camera during the gameplay to ease gamers' control over their character. These approaches are usually limited to one target and have no consideration for cinematographic quality. Haigh-Hutchinson detailed a large spectrum of these techniques in [HH09].

2.2.3 Offline camera planning

Even though a large part of the literature tries to address camera planning as an online process, other approaches tackle the problem with no regard to any real-time constraints. Since most of these solutions are not dedicated to video game camera control or any other interactive application, the computational time required to generate a solution is not a major concern.

A recent contribution by Lino and Christie [LC15] demonstrated the use of their camera placement solution [LC12] for camera path planning. Their system enables the creation of complex camera motions such as long takes using a limited set of keyframes to specify camera composition at precise moments in time. To compute a path between two keyframes, they first compute the two camera configurations and then express each of them in the other's toric space. By using interpolation of the parameters in the toric space of the two cameras, they are able to generate two separate camera paths that can be blended together (cf. Figure 2.17).

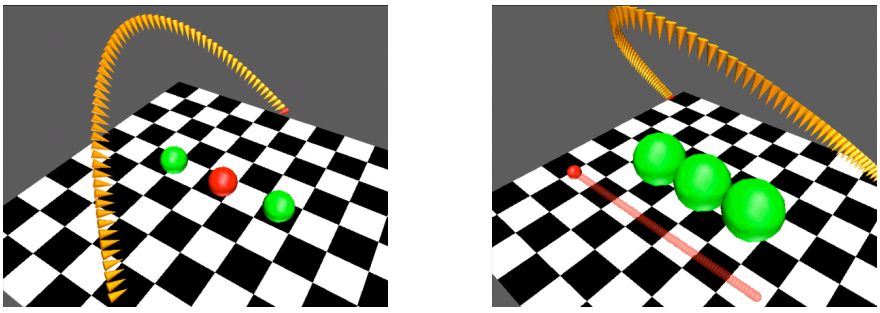


(a) **(b)**

Figure 2.17: *Composition-based interpolation of the camera position around a pair of targets A and B (a). Interpolation curves over the camera motion and re-framing along time (b).[LC15]*

In section 2.2.1 we mentionned CAMPLAN [OHPL99, HO00] as an optimization system to compute static shots. However, the system was also designed to plan camera motion for

both static and dynamic scenes (cf. Figure 2.18). By considering the path of a camera to be a quadratic bezier curve between two known points (established using the static version of CAMPLAN), a camera path is found by optimizing the control points to maximise visual properties specified by a user. Samples of camera positions along the path are evaluated, and the cumulative fitness of the path is estimated as a linear combination of the fitness at the sampled positions. The system however does not handle polygonal worlds ; it is limited to spheroid approximations of the scene elements.



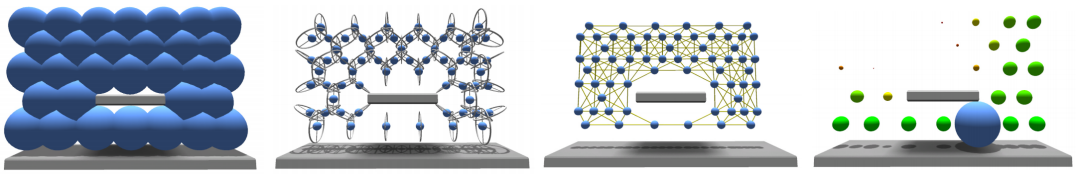
(a) **(b)**

Figure 2.18: *Camera planning with* CAMPLAN *[OHPL99, HO00] on a static (a) and a dynamic (b) environment.*

Dedicated to the specific task of creating overviews of human motions (*e.g.* from mocap data), [ACOYL08] cast the problem of camera control as an energy minimization process guided by the conjunction of external forces (describing the viewpoint quality at each location and time) and internal forces (enforcing smoothness on the generated path). As a result, the system generates a sequence of camera paths (with edits) using an optimization process on a potential field defined by the aggregation of forces. While purely based on character motion analysis and viewpoint quality heuristics, the process generates meaningful overviews of human motions without integrating semantic aspects. Later, [LLY*12] proposed to automatically extract *social events* from the joint analysis of multiple character motions, e.g. related to trajectories and similarities in distance and direction of characters' motion. Events are then processed, analysed for spatio-temporal correlation and ranked so as to generate motion clip segmentation. This segmentation is used as a basis to express an optimization problem on the camera parameters for each motion clip (actually long motion clips can be separated into segments solved individually). Following the lines of [ACOYL08], the optimization process is guided by internal, external and continuity forces shaping a potential field. Continuity forces integrate the 180-degree rule as well as the jump-cut rule. The computational cost of the optimization process remains significant (1 to 3 minutes).

Recently, in [OSTG09], Oskam *et al.* proposed a hybrid solution that can run in real-time after pre-processing the environment. The system performs a pre-computation of visibility between all possible locations in a static environment, and create a visibility graph that encodes this information. When tracking a single target (which the method is limited to), the camera relies on the visibility graph to plan the best path towards the target (following different criteria). Changes in environments are partially handled by dynamically updating the visibility graph at the expense of many visibility tests. Figure 2.19 illustrates the creation of the roadmap by going through every steps of the algorithm.

Finally, part of the research on camera control addresses the challenge of scene exploration in virtual environments. Even though those virtual walkthroughs mostly use a single camera



(a) **(b)** **(c)** **(d)**

Figure 2.19: *Overview of the creation algorithm for the roadmap [OSTG09]. A spatial discretization is computed from an initial geometry of the environment (a). Portal regions are computed for overlapping spheres (b). A graph is then built from the portal regions (c). Finally, for each pair of spheres, a visibility probability is computed with a Monte-Carlo raytracing algorithm (c).*

and their direct purpose is not telling a story, they offer interesting approaches to camera control. In [BDP99, BDP00] for instance, the authors proposed a solution to move the camera on a sphere surrounding the scene by using several heuristics to find the best path on this surface. In [AVF04], the authors introduced *Way-Finder*; this system generates guided tours of complex walkthrough models. Their approach relies on identifying the free-space structure of the scene (represented by a cell and portal graph) and an entropy-based measure of the relevance of a view-point. This metric is used for deciding which cells have to be visited and for computing critical way-points inside each cell. In [VS03], Vásquez *et al.* focused on automatic indoor scene exploration based on Information Theory. With their approach, the camera path is guided by viewpoint entropy to always provide the maximum amount of new information. Their system however remains limited to three possible camera movements: turn left, turn right and move forward. Later, Serin *et al.* [SAB12] also used the concept of viewpoint entropy for best view determination to address the challenge of terrain navigation. In [SP08], the authors introduced the notion of semantic distance between objects of the scene. After choosing a set of good viewpoints that gives users a maximum knowledge of the scene, they compute a camera path between them that account for the semantic information instead of only using the geometry of the environment.

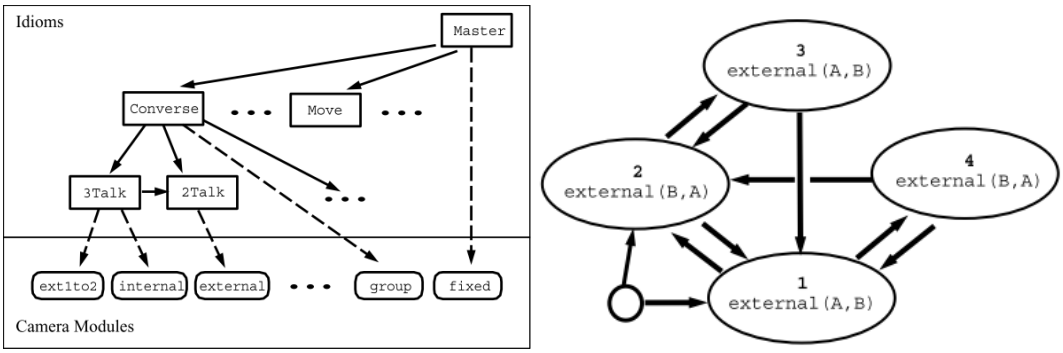
2.3 AUTOMATIC FILM EDITING

Automating the editing process is a really complex challenge as the editing remains strongly dependent on the director's style and point of view. So far, several approaches have offered interesting solutions to this difficult task. In [Ron12], Ronfard reviews some of the existing solutions and categorizes them under three main categories : procedural, declarative and optimization approaches. In this section we use a different classification. We first review the largest part of the research on the field that rely on cinematographic idioms (section 2.3.1). We then present some work based on optimization (section 2.3.1) and finally review other alternatives that presses the narrative aspect of editing

2.3.1 Idiom-based editing

Starting in 1996 the *Virtual Cinematographer* [HCS96] first formalized the notion of film-idioms (mentioned in section 2.1.4) in the context of virtual cinematography. Film idioms are "recipes for obtaining good framing and editing in any given situation, similar to cases in case-

based reasoning" [Ron12]. Solutions based on film idioms are close to live-cinematography as they try to imitate and simplify the process by combining director and editor: decisions can be made on the fly. The Virtual Cinematographer implements this cinematographic knowledge (idioms) through a set of small hierarchically-organized finite state machine (FSM). Each idiom is composed of a set of camera modules associated with the states of the FSM. A camera module takes as input the number of actors. It then automatically positions and orients the camera to place actors at particular locations on the screen. It also accounts for editing rules in the placement of the cameras (such as the 180-degree rule).



(a) *Virtual Cinematographer structure* **(b)** *FSM of a two-character dialog idiom*

Figure 2.20: *The Virtual Cinematographer structure is hierarchically composed of idioms (a). Idioms are encoded as FSM where each state is associated to a camera module (b).*

To test their results on a simple "virtual party", the authors implemented several idioms (Moving, Converse, 2Talk and 3Talk) as well as sixteen distinct camera modules. While this system presents the advantages of being simple and running in real-time it is still limited by the heavy task of defining the idioms. Indeed, as they are hard-coded, the idioms still require expert knowledge from the user. Moreover, the use of deterministic finite state machine results in a lack of expressiveness as every similar situations will be handled with the same idiom.

In the same time, [CAH*96] introduced the *Declarative Camera Control Language* (DCCL) as a general framework for generating idiom-based solutions to cinematography and film editing problems. The language devised by Christianson *et al.* is used to formalize idioms using four primitive concepts: fragments, views, placements and movement endpoints. Shots are composed of a collection of one or more fragments defining an interval of time during which the camera is either static or operating a simple motion. An idiom is defined by a name, a list of arguments, a list of shots described in the DCCL format and by specifying the activities it should be associated with. The system presented in the paper relies on a *Camera Planning System* (CPS) to generate the final movie from the specified idioms. The CPS takes as input an animation trace (*i.e.* geometric and narrative information about the scene) and uses a *Sequence Planner* to build a film tree (see Figure 2.22) with all candidate idioms for each scene of the sequences in the movie. The idioms (expressed in DCCL) are then compiled to place and move the cameras in the actual environment, generating all candidate frames. Finally a heuristic evaluator selects appropriate candidate frames for each scene.

This solution suffers from the same inconvenient as the Virtual Cinematographer. Even though it allows more expressiveness thanks to the use of the DCCL, such approach fails to be extensible due to the burden of creating new idioms for each style, action and context. Moreover, it cannot be considered fully automatic as it still requires expert knowledge for the creation of idioms. Finally, despite having proved efficient to create cinematographically

Figure 2.21: *Pipeline of the CPS presented by [CAH*96]. After finding all candidate idioms from the simulation data using a sequence planner, the CPS compile them to generate all candidate frames. Finally, the candidate frames are evaluated to compute the best possible edit.*

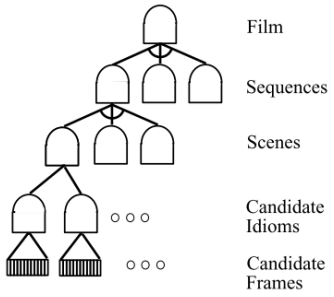


Figure 2.22: *Film tree expanded by the CPS of [CAH*96]. Each scene of the sequences of a movie can be filmed through one or several candidate idioms. Candidate frames are computed in the virtual environment for each idiom using the DCCL compiler.*

correct edits on simple cases such as dialogue scenes, this approach still lacks the ability to account for directors' specific communicative goals.

The seminal work of [CAH*96] and [HCS96] has been extensively used and extended by many researchers. Markowitz *et al.* [MKSB11] for example extended the film idiom approach by replacing finite state machines with behavior trees. Using behavior trees allows more flexibility as it is possibly non-deterministic. Unlike previous solutions based on FSM, the system is able to handle any obstacle that arises and avoid getting stuck in a state when encountering unexpected events. It also enforces the 180-degree rule and the 30-degree rule, assuming a single line of interest at any given time in the scene. Similar to film idioms, their solution remains reactive, resulting in an editing style that immediately follows the action and cannot easily control the pacing or style of the editing.

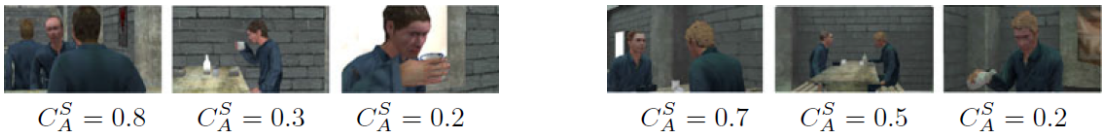
Friedman and Feldman [FF04] presented a knowledge-based camera planner (namely *Mario*) which encodes both cinematographic and film editing knowledge, in a similar way to idioms, but in the form of rules. Using a knowledge acquisition process they converted domain expert principles taken from well-known textbook (such as [TB93]) into declarative rules. Then, to compute the sequence of shots, they use a non-monotonic reasoning process. As a result, film idioms are discovered heuristically as solutions obeying the rules in a particular situation. The main issue with their solution is that the reasoning is exponential in the number of film frames and cameras. Moreover, the system was only tested on simple solutions with limited number of actors and actions (always sequential).

More recently, Amerson and Kime devised a system for real-time camera control in interactive narratives called FILM [AKY05]. Their work strongly relies on [CAH*96] and [HCS96] foundations. They implemented their own *FILM language*, similar to DCCL, that also accounts for the narrative impact of an idiom. The generated film tree is hierarchically organized so that each subsequent level contains a scene that is more refined and conveys more information to the user. Their system is managed by a director that receives narrative information and perform a depth first search on the tree to find the most appropriate idiom to convey the narrative dis-

course. The director then sends directives (idioms) to the cinematographer which is in charge of the rendering. The idioms are expressed in the *FILM language* as weighted constraints. In case of an over-constrained problem, the system uses a procedural model for the relaxation of constraints. Even though this work offers an interesting alternative to [CAH*96] and [HCS96] – it runs in real-time and offers the flexible structure of film trees – it still has the main inconvenient of the two methods: it requires the creation every idiom for every situation. Furthermore, the implementation of this approach was not completed and thus remain untested.

2.3.2 Optimization-based editing

Another approach consists in considering film-editing as an optimization problem. In [LCCR11] and [CLR12], the authors presented a framework capable of generating cinematics for interactive storytelling. Their system uses a path finding algorithm that runs in real-time to find an approximate solution. It chooses shots and cuts incrementally as the story unfolds. To decide whether to stay within the current shot or perform a cut, the system relies on simple heuristics based on cinematographic and editing rules mostly implemented as binary cost functions (see Figure 2.23). This work offers interesting perspectives yet remains limited by its realtime constraints and the binary aspect of its cost functions.



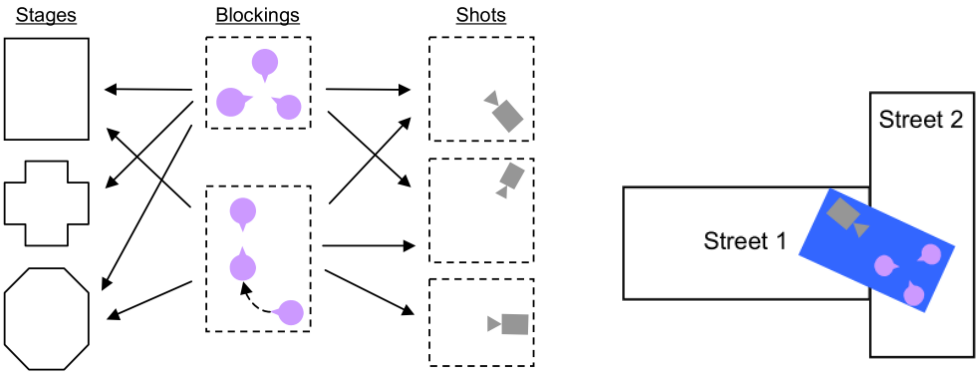
(a) **(b)**

Figure 2.23: *Illustration of shot costs computed by [LCCR11] to find the best solution: (a) three shots of a drinking action ; (b) three shots of a pouring action*

With the Cambot system, Elson and Riedl [ER07] also addressed the challenge of automatic editing using an optimization-based approach. Unlike standard idiom-based solutions, they encoded three distinct layers (or facets) of cinematic knowledge (see Figure 2.24): *the stage* (area of space that the system assumes to be free of occlusions and obstructions), *the blocking* (placement and movements of the characters within the stage) and *the shot* (position, orientation and focal length of a virtual camera relative to the stage ; it also handles camera motion). From a given scenario (script Figure 2.25a) and a set (virtual environment), the system automatically computes the stages, blocking and shots from an hand-authored library of facets. With the correlation between the three facets they ensure the quality of the generated rushes (actors were placed to avoid occlusion or obstruction).

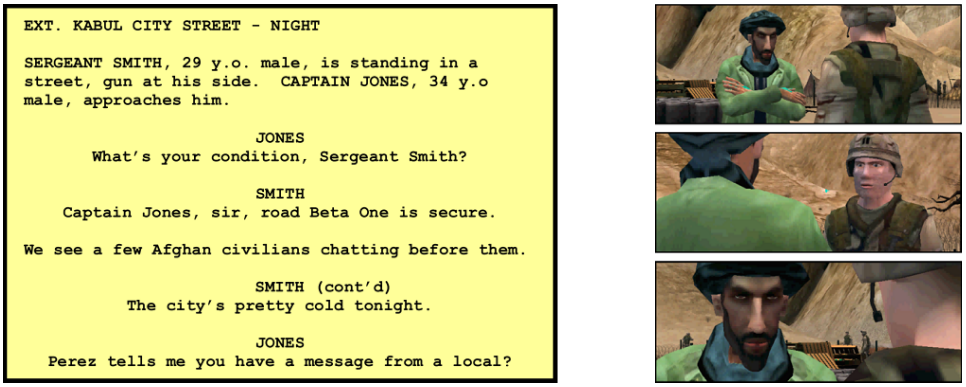
With a Markov assumption, the editing of the rushes is then computed using a dynamic programming algorithm (see Figure 2.26) which searches for the sequence of shots that best covers the beats. With their solution, only one shot can be selected for the whole duration of a beat ; transitions are only considered between beats.

As promising as the system looks, it still presents many limitations. First of all, its expressiveness is limited by the assets available in the facets' library. Moreover, it requires the scenario to be decomposed into a linear sequence of beats (*i.e.* it can not deal with overlapping actions and dialogues). Even though the beat-by-beat structure allows to considerably reduce the search-space, it also prevents any precise control over the pacing. As the cuts can only be operated between beats, it is not possible to choose the precise cutting point between shots. Furthermore the system was built to handle every aspect of the movie-making process (creating



(a) **(b)**

Figure 2.24: *This figure shows the correlations between the three types of facets (a) and how they can be arranged in a set (b). The blocking of the characters is contained within the stage so that no occlusion or obstruction can occur. And the cameras are placed at predefined positions to shot the sequence under stereotypical viewpoints. [ER07]*



(a) *Scenario given as input to the system* **(b)** *Final rendered movie*

Figure 2.25: *The cambot system [ER07] takes as input a scripted scenario (a) and generates a fully animated movie (b).*

the stage, placing the actors, shooting the scene and editing it). Taken separately, the editing step might not generate pleasing results. For instance, it could not be applied as it is on an already existing animation: the quality of the shots would not be ensured as the system would not control the characters' placement. Finally, the paper does not provide any detail on the selection process during the editing. The heuristics used to cut from one shot to another are not described.

A similar approach was later used by *Xtranormal Technologies* as part of their Text-to-Movie animation pipeline. Their *Magicam* system [Ron10] also computes the editing of an animation movie as a solution to an optimization problem. But unlike *Cambot*, it represents film editing as a search problem over a space of semi-Markov chains, allowing to control the rhythm of the editing while satisfying the rules of film editing. However, *Magicam* is a proprietary system whose internal details are not disclosed, and whose validity is not demonstrated.

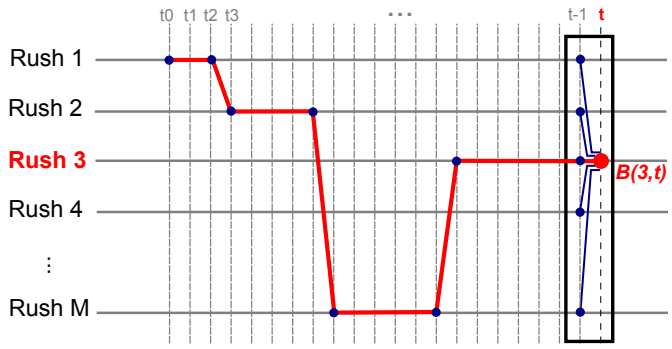


Figure 2.26: *This figure illustrates a dynamic programming approach to the problem of film-editing. The best edit at time t is computed only from the best solutions at time* t-1 *and the transitions between rushes. Cuts can only be performed between narrative beats (blue dots).*

2.3.3 Narrative-driven editing

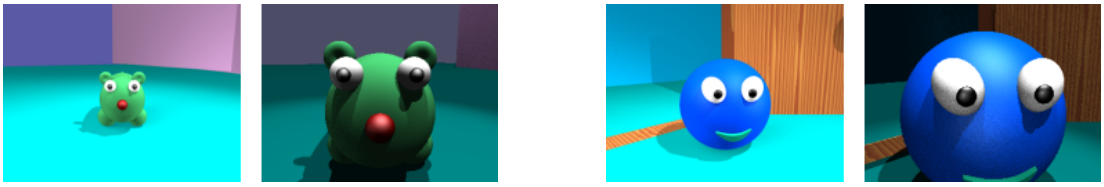
Most of the work previously mentioned aims at producing a finished movie that follows cinematographic and editing standards. At the exception of [AKY05] (that was not implemented), none of the proposed models succeeded to really account for both the narrative impact on the audience (to convey emotion for example) and the cinematographic quality. They lack considering the viewer's comprehension of the complete sequence of shots and cuts. The following related work presents a last class of editing methods that attempt to drive the editing from narrative goals.

Tomlinson *et al.* [TBN00] proposed a fully automatic method for generating expressive cinematography. Their system uses autonomous cameras, with emotional states and goals, which choose between a number of visual targets – such as a specific character, two characters interacting with each other, or one character interacting with its environment. Though their system interestingly motivates the shots both emotionally and visually, the rules of editing are not enforced, and the pacing of the shots is resolved by ad hoc rules.

Kennedy and Mercer [KM02] directly addressed the problem of planning a complete sequence of shots for a given sequence of actions which must be known in advance. Users can choose between fast or slow editing in two different styles. Authors used a depth-first forward chained planner which can only evaluate a small number of possible plans and enforce a single continuity editing rule ("not crossing the line of action"). The system uses the LOOM knowledge representation language to encode different communicative acts in the rhetorical structure theory. By mapping the story-line into communicative goals, stated in terms of themes and moods, the system is able to plan the choice of camera and perform the editing. Figure 2.27 illustrates how a simple zoom along with a subtle change in lighting can affect how a character or an event is perceived.

With their Darshak system [Jha06, JYM10, JY11], Jhala and Young propose an AI-based approach to virtual cinematography that relies on a hierarchical partial order planner. Taking as input a structured representation of the sequence of actions in the scene, the system searches for the best idioms and best shots to convey the actions. Camera shots are defined as intentional communicative acts planned by the camera planner to influence the beliefs and mental states of the viewer. The system relies on a hierarchical partial-order causal link (POCL) planning algorithm to generate story events and directives for filming them.

In parallel, [CJBY08, DYR11b, DYR11a] addressed the specific problem of automatically generating cinematic highlights for game sessions. The authors of [DYR11b, DYR11a] pro-

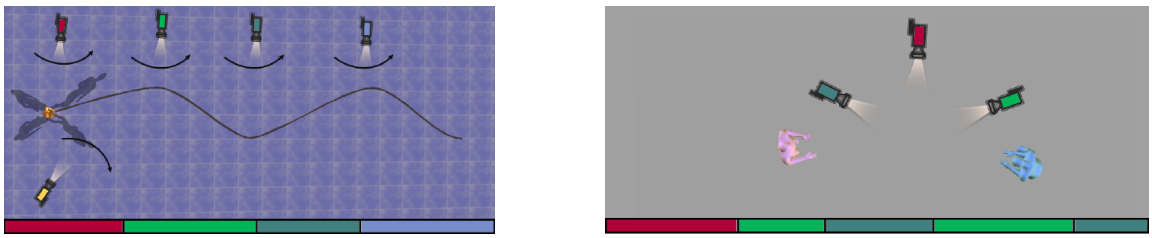


(a) **(b)**

Figure 2.27: *A change in shot size along with a subtle change in lighting can affect the audience's perespective [KM02]. (a) The character that seems happy in the first frame looks scary in the other one. (b) The dramatic tension between the two frames is significantly different.*

pose Afterthought, a system that analyses actions performed by the players to recognize narrative patterns expressed as Finite State Machines. The detected narrative patterns are then paired with cinematographic instructions to generate a meaningful cinematic sequence. Cinematographic instructions then trigger camera scripts in the rendering environment to compute viewpoints, taking care of visibility and optimal distances to characters. Interestingly, the proposed approach relies on patterns of actions to build the cinematographic instructions, in a principled and context-sensitive way. The mapping is however straightforward which leads to the repetition of a cinematic sequence with two identical patterns.

Finally, even though one might question the relation between their work and storytelling, Assa *et al.* tackled an interesting challenge that is the task of creating overviews of human motions (*e.g.* from mocap data). Their narrative goal here is to convey as best as possible human motion in an animated sequence. In a first paper [ACOYL08] they presented a solution relying on camera control and targeting only one character. Later, in [AWCO10], they propose a fully automated editing process that performs cuts in real-time between multiple cameras viewing human motions. The ranking between shots is computed by measuring for each camera the correlation between human motions in the 3D scene, and the on-screen 2D projection of these motions (the larger the correlation, the better the shot). A measure of erosion of the current view is employed to motivate the cuts between the viewpoints, while enforcing continuity rules (jump-cut and crossing the line of action). Figure 2.28 gives examples of camera set-up for two different animated sequences. Despite being restricted to viewing human motion their results have shown that in simple cases, their solution mimics the behaviors of cinematic idioms.



(a) **(b)**

Figure 2.28: *Camera setups for a long character walk (a) and a conversation-like scene (b). The timelines at the bottom of the images indicate the selected camera as the animation is being played. [AWCO10]*

2.4 SUMMARY

In this chapter, we covered a large spectrum of the existing techniques that address the problems of virtual cinematography and editing. Through this state of the art, we observed the lack of existing methods capable to coordinate multiple cameras. We also noticed that real-time approaches to camera control currently provide poor cinematographic quality and do not consider narrative goals. Finally, no existing techniques gave any consideration whatsoever on the realism of the motion of the cameras. In this thesis, we address each of these limitations through different techniques presented in chapters 3, 5 and 6.

In terms of editing we highlighted the need for more automatic methods. Existing solutions based on idioms still require too much hand work. Optimization based techniques on the other hand rarely focus on the narrative aspects of editing or do so at the expense of the cinematographic and editing quality. Chapter 4 presents an optimization based solution that ensures the cinematographic quality while using narrative information to drive the editing.

CHAPTER

3

STEERING BEHAVIORS FOR AUTONOMOUS CAMERAS

The automated computation of appropriate viewpoints in complex 3D scenes is a key problem in a number of computer graphics applications. In particular, crowd simulations create visually complex environments with many simultaneous events for which the computation of relevant viewpoints remains an open issue.

In this chapter we address the challenge of controlling multiple cameras in a dynamic environment. We present a system which enables the conveyance of events occurring in complex crowd simulations. It relies on Reynolds' model of steering behaviors to control and locally coordinate a collection of camera agents similar to a group of reporters. The key benefit, in addition to the simplicity of the steering rules, holds in the capacity of the system to adapt to the evolving complexity of crowd simulations by self-organizing the camera agents to track interesting events.

3.1 INTRODUCTION

Crowd simulations generate complex 3D scenes that include both local events (individual actions of crowd members) and larger scale events emerging from many individual behaviors such as a group of individuals simultaneously moving from one location to another. Conveying the resulting animations to spectators is therefore a challenge. Unfortunately, as seen in chapter 2, previous realtime camera control methods focus on maintaining the visibility either of a small number of targets (see [HHS01, OSTG09, CNO12]), or of a large number of targets but from a single viewpoint [VMGL12]. Other methods are designed for highlighting the motions of isolated characters or of predefined groups of characters [ACOYL08, LLY*12, AWCO10], or focus on high-level aspects by encoding elements of cinematographic rules which are only applicable to a small number of characters (see [LCL*10]). Closer to our approach, the method proposed by [BJ09] also relies on a force-based system. However, the use of APF to find the optimal camera placement is not appropriate in the case of tracking large group of targets. The camera might never find an optimal solution, resulting in continuous jittering motion. Moreover, their method does not implement cinematic camera movements. Therefore, capturing a full crowd simulation, with the variety of multi-scale events taking place, calls for the design of new methods.

In this chapter, we propose a novel approach for controlling multiple cameras, in the task of conveying as-many-as-possible of the events occurring in a crowd simulation. The original idea is to simulate a team of reporters trying to capture the diversity of behaviors in an environment encompassing both local events (interactions between small numbers of characters) and emerging events that involve many characters. Our approach for animating cameras draws its inspiration from Reynold's steering behaviors [Rey99]. Indeed, each camera is animated separately as an autonomous agent in our system. This ensures smooth displacements and rotations of cameras. Our camera behaviors are driven by the necessity to both convey local and emerging events, while ensuring a proper distribution of camera agents around the events. More precisely, our camera agents automatically avoid static objects that obstruct the visibility of targeted events. In addition, camera-agent behaviors are specifically designed so that each of them either scouts the environment, searching for new events to shoot, or tracks existing events. While tracking an event, the steering behavior favors good views of the event. When scouting for an event, the steering behavior favors events that are not already followed by other camera agents. We also set up specific interaction rules between camera agents, preventing them from tracking the same event from similar viewpoints. Nearby camera agents are steered away from each other both in position and in orientation. Distant camera agents following the same event are also steered to rotate around the event and take different views of it. All those desirable properties are obtained by allowing the camera agents to be steered horizontally (as in Reynolds' original paper) and also vertically and rotationally (with pan and tilt angles). As a result the user is therefore provided with a gallery of relevant shots on individual and emerging events occurring in crowd simulation, among which the user can select the appropriate animation sequences.

The first key benefit of this approach stands in its simplicity. We show that a small range of simple steering behaviors are sufficient for creating quality camera motions, while covering a wide range of the events occurring in the simulation. Behaviors are crafted in a way which is independent of the crowd simulation model. The second benefit stands in the adaptivity of the method: the same set of steering behaviors enables camera agents to adapt to a wide range of situations, from a small number of cameras in a simulation with many occurring events to a large number of camera agents with a small number of events. In crowd simulations, in which

individual and emergent behaviors of the characters are mostly unanticipated, those benefits are especially important.

3.2 Background on Steering Behaviors

In this section, we review some important concepts related to steering behaviors. Some of them will be extended to the case of steering cameras in Section 3.3.

3.2.1 Agent dynamics

In Reynolds' approach, autonomous agents are driven by steering forces [Rey99] and their position is updated at every time step in the simulation as follows:

Algorithm 1 Agent dynamics. At each time step, the acceleration of an agent i at time t (denoted $\gamma_i(t)$) is computed as a sum of forces, and a Euler integration is performed to compute velocity $v_i(t)$ and position $p_i(t)$.

$t = 0$
while simulation is running **do**
  for all agents i **do**
    Update steering forces $F_{ij}(t)$
    $\gamma_i(t) = \sum_j F_{ij}(t)$
    $v_i(t) = v_i(t - \delta t) + \gamma_i(t)\delta t$
    $p_i(t) = p_i(t - \delta t) + v_i(t)\delta t$
  end for
  $t = t + \delta t$
end while

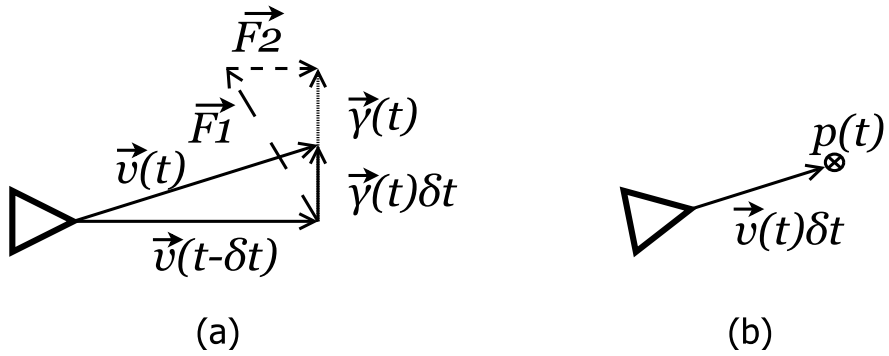


Figure 3.1: *Computation of forces applied to an agent. The agent is represented as a triangle: (a) the agent's velocity $v(t)$ is updated by integrating acceleration expressed as a sum of forces; (b) the agent is then oriented along the newly computed velocity $v(t)$.*

3.2.2 Steering forces

Simple steering forces enable the creation of complex simulations with emerging behaviors. A simulation is created using a set of agents that are defined by a set of characteristics: position, velocity, radius, mass, maximum force and maximum speed. Motions of the agents

are generated using these simple characteristics, and by applying a simple particle based algorithm where each particle is represented as a "vehicle". In order to improve the efficiency of the entire computation and implementation of the system, Reynolds specified a simple but efficient constraint: agents always move in their forward direction. Despite this simplifying assumption, Reynolds was able to present a rich vocabulary of steering behaviors driven by specific steering forces including seeking, fleeing, pursuit, evasion, offset pursuit, arrival, obstacle avoidance, wandering, path following, wall following, containment, flow field following, unaligned collision avoidance, separation, cohesion, alignment, flocking and leader following.

Many of these different forces are based on a simple concept: the force that will be applied to the agent is computed by subtracting the current velocity of the agent from the desired velocity related to the behavior. The acceleration being the sum of all the forces, at each time step, the velocity will be updated by adding part of the acceleration (depending on the time step) which will reduce the difference between the velocity and the desired velocity. The seeking force devised by Reynolds and illustrated in Figure 3.2(a) is based on this concept. The desired velocity is computed by first finding the direction vector, and then multiplying it by the maximum speed of the agent. The direction of the desired velocity is the vector between the target and the agent.

For the sake of clarity, we hereby illustrate the design of the arrival behavior: an agent should slow down when reaching its target. When the distance between the agent and the target becomes smaller than a threshold distance, the desired velocity needs to decrease with the distance. Therefore, it is multiplied by the ratio of the distance to the threshold distance. When the distance is greater than the threshold distance, the normal seeking force is applied. Figure 3.2(b) illustrates this arrival behavior.

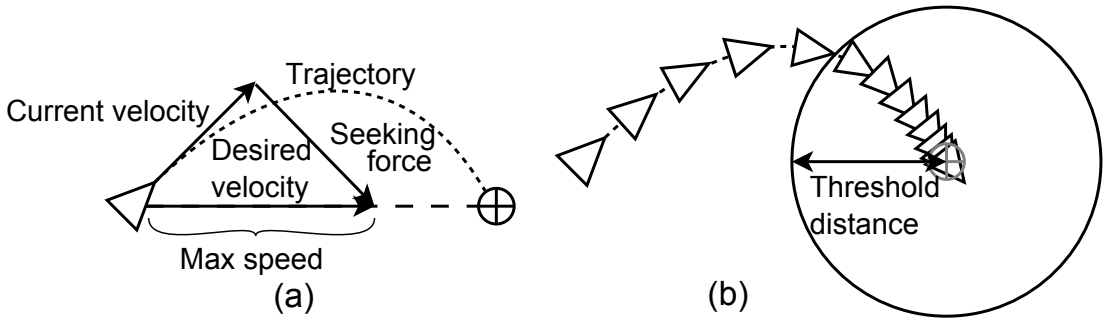


Figure 3.2: *Illustration of two behaviors: (a) seeking behavior, (b) arrival behavior*

3.3 STEERING CAMERAS

In this section, our objective is to implement appropriate steering behaviors for our camera agents. The task assigned to camera agents is to convey individual and emerging events occurring in a crowd simulation. For this purpose, we propose two steering behaviors:

- **Scouting**: default behavior when a camera is not following an event; the camera is searching the scene for events to track.
- **Tracking**: behavior of a camera while it is following an event; the camera is in recording mode during tracking.

In both cases, we need to extend the dynamics of Reynolds' agents to desynchronize the agents' direction of motion from their orientation (a strong hypothesis proposed by Reynolds). We will explain how this extension is performed in 3D with cameras that can change their pan and tilt angles independently of their position and motion direction, while simultaneously changing their elevation. We will then describe camera-specific forces and torques, whose combined effects generate the desired camera behaviors during scouting and tracking.

3.3.1 Targets and events

We define an *event* in the simulation as a spatio-temporal segment where a particular behavior is observed, should it be relative to an individual behavior, a one-to-one interaction behavior or a group behavior (*e.g.* flocking, herding, and leader-following, where an entire group of characters follows a recognizable and coherent motion pattern). Each event involves a number of characters (characters concerned by this event in the crowd simulation). Our representation is detached from any specific crowd simulation model but requires a step to extract the events from the simulation using geometrical features relative to the characters.

More precisely, the computation of events is based on the following information:

- position and speed vector of the characters;
- orientation of the characters (forward vector).
- type of the characters (if the simulation uses several types of characters);

In this work, we propose to extract two types of events:

- *one-to-one interaction*: two characters face each other for a long time without moving (see Figure 3.3(a));
- *group motion*: emergent behavior observed when a group of characters is moving together in the same direction (see Figure 3.3(b)).

Figure 3.3: *During scouting, cameras search for interesting events: (a) One-to-one interaction event; (b) Crowd behavior event.*

Our *one-to-one interaction* events are easily detected when pairs of characters face each other with zero velocity. Our *group motion* events are detected by studying a group of characters, based on their direction, speed, density and homogeneity in the following way:

- Average direction: by summing the normalized speed vectors of each character and then dividing the total by the number of characters in the group, we get a vector which magnitude will be related to the alignment. Thus the length of this vector gives us a good estimate of the alignment of the characters in the group: the closer the value is to one, the more the group is aligned.
- Average speed: The speed of each character is closer to the average speed.
- Density: the amount of characters per unit of surface.
- Homogeneity: the number of characters in the group belonging to the same type.

In order to efficiently implement the detection of group motions, we rely on a quad-tree representation. The scene is divided into a 2D multi-resolution grid and, starting from the finest (highest) resolution, we evaluate in each cell whether the characters in the cell form a group motion. At the next (coarser) level, we create a group containing all cells with similar group motions and repeat the process until it reaches the coarsest level in the hierarchy. During group motion, both the group and its motion are continuously updated over time. New group motion events are created when a new group is detected and deleted when the group becomes too small.

3.3.2 Camera dynamics

In Reynolds paper, one simple assumption is made: the agents are always moving in the direction of their orientation, *i.e.* their velocity vector (that gives the movement direction and the speed) and their forward vector (gives the orientation of the agent) are aligned. Implementing steering behaviors for a camera agent however requires to distinguish the camera orientation from its direction, typically moving in one direction while filming in another one.

Simple strategies that consist in re-orienting the camera at each time step towards the center point of the group of characters involved in an event would fail. The varying number of characters in an event would make the center move significantly at each time step, leading to undesirable jerky motion. Moreover, no targets are available in the scouting behavior (since no events are detected). Instead, we propose to apply a dynamic model on the camera orientation, expressed as a torque. Camera position and orientation are therefore computed separately: steering forces are applied to move the camera around the scene and rotation forces are applied to turn the camera around. The model intrinsically produces smoother changes in orientation.

Figure 3.4 illustrates the computation of a new camera orientation given different forces (in 2D for illustration). First, the angular acceleration is computed using the torques. Then, the rotational velocity is updated using the acceleration. Finally, the new orientation is computed with the angular velocity. A camera agent is represented by its 3D position in the Cartesian space p_i, a pan angle θ_i and a tilt angle ϕ_i. For the sake of clarity, we denote the camera orientation as a quaternion q_i and rewrite the camera dynamics equations using quaternion multiplication. The algorithm 2 details the computation of the new orientation used at each time step.

3.3.3 Camera steering forces

We now detail the design of steering forces associated to each camera. The essential requirements are: maintaining a good framing of an event, maintaining visibility of an event, keeping a

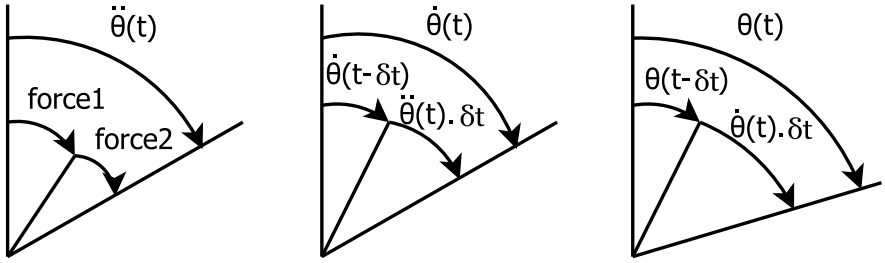


Figure 3.4: *2D representation for the computation of the camera's horizontal orientation (θ) from the angular acceleration ($\ddot{\theta}$) and the angular velocity($\dot{\theta}$).*

Algorithm 2 Camera dynamics: $\ddot{q}_i(t)$ represents the rotation acceleration of agent i at time t, $\dot{q}_i(t)$ the rotation velocity and $q_i(t)$ the orientation.

while simulation is running **do**
  for each camera agent i with orientation q_i **do**
    Update steering forces $F_{ij}(t)$ and torques $T_{ij}(t)$ of agent i
    Update camera position as usual
    $\ddot{q}_i(t) = \sum_j T_{ij}(t)$
    $\dot{q}_i(t) = \dot{q}_i(t - \delta t)\ddot{q}_i(t)\delta t$
    $q_i(t) = q_i(t - \delta t)\dot{q}_i(t)\delta t$
  end for
  $t = t + \delta t$
end while

given distance to an event and finally ensuring that different cameras cover different viewpoints of the same event.

In order to control the camera position, five forces have been designed: target following, obstacle avoidance, camera separation, wandering and containment. We now review them one by one.

Framing force:

moves the camera towards the closest optimal camera position in terms of framing (composition of characters on screen (without considering obstacles). The goal of this force is to ensure that the camera maintains a specific framing of an event, *i.e.* that characters composing the event stay within the camera frustum. To ensure this framing, we designed a force which attracts the camera agent towards a range of viewpoints defined by two points A and B. This range represents a continuous set of viewpoints (see examples in Figure 3.6) for which the framing can be easily computed. The computation relies on Lino and Christie's frame composition technique [LC12] by considering that a group of characters can be framed by framing their left-most and right-most characters on the screen as illustrated in Figure 3.5(a).

Indeed, as seen in chapter 2.2.1, for two points A and B, viewed with a constant angle α, the camera must lay on a circle of radius 2α [LC12]. Figure 3.5(b) shows the two arcs representing the possible positions of the camera. In our case, we choose the angle α as a percentage of the field of view and then compute the two arcs.

However, although all cameras positioned on these two arcs maintain a constant viewing angle, the viewpoints which are too close to A or B (displayed in red dashed lines in Figure

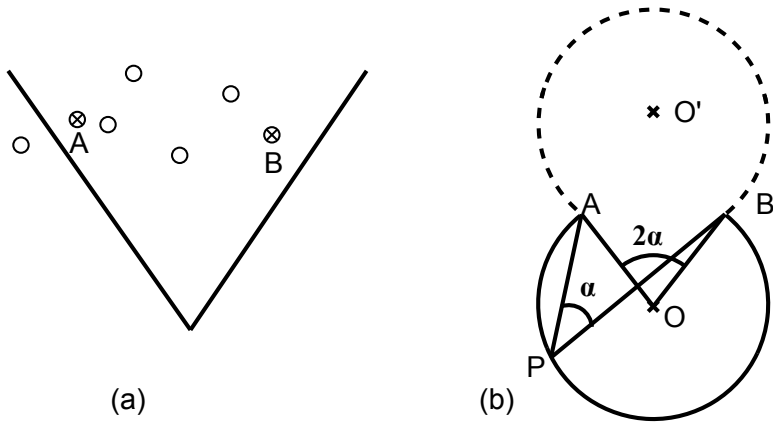


Figure 3.5: *Framing force: (a) selecting the left-most and right-most characters of a group* w.r.t *the current camera position enables (b) the construction of a specific range of viewpoints, for which the framing of both characters is ensured.*

3.6) are not suitable. Indeed, when framing a group of two characters we prefer to avoid these less informative shots (very close shots to one or the other agents). To account for this problem, we propose to modify Lino and Christie's approach by changing the range of possible camera positions when the angle $(\vec{O'P}, \vec{O'O})$ is less than a minimum value (β), meaning the camera is too close to the agents (see Figure 3.6). We rely on a Bezier curve to define this range. The three control points P0, P1 and P2 are computed by using the values of β, and P1 is the intersection of the tangent lines to the two circles in P0 and P2.

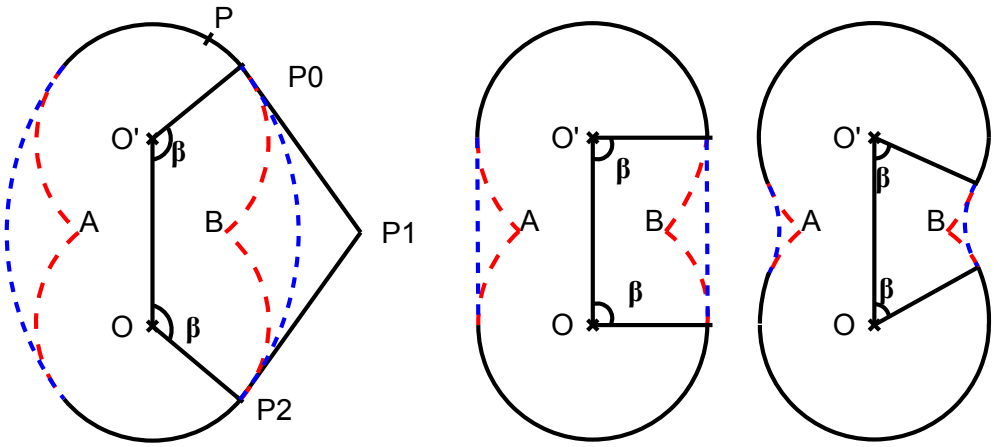


Figure 3.6: *Framing force: modifying the range of viewpoints around two characters A and B, in function of β. The framing force pushes the camera to the closest point on this range.*

The force is then easily expressed by using Reynolds' arrival force targeting the closest point from the camera agent to the range of possible viewpoints. In addition, when the camera is filming a large group, a force elevating the camera is added to mimic crane cameras and to improve the visibility of the agents.

Obstacle avoidance force:

moves the camera agent on one side or the other depending on the position of the obstacle in the frustum. This force differs from the obstacle avoidance force presented in Reynolds' paper: the process loops on the list of obstacles entirely or partially included in the frustum. Each time an obstacle at position o is in the frustum and is closer to a camera agent position p_i than a target position k is, we consider the obstacle may potentially occlude the target. To prevent these potential occlusions, a force is applied to the camera depending on whether the obstacle is in the left part of the frustum (in such case the force moves the camera on the right) or in the right part of the frustum (in such case the force moves the camera on the left). The position of the target k is computed as the center of all characters involved in the event. For each obstacle, the force is computed by subtracting the current velocity v_i of camera agent i from a desired velocity. See Algorithm 3 for details.

Algorithm 3 Obstacle avoidance: computes a sum of forces F_{obs} that pushes the camera on the left or the right according to the relative positions of the obstacles and the target. l_i represents the normalized look at vector (orientation) of camera agent i at time t, r_i represents the normalized right vector of camera agent i at time t and v_{max} is the maximum allowed velocity for the camera.

for each obstacle at a position o in the frustum of camera agent i **do**
  // check whether the obstacle is closer to the
  // camera than the target is
  if $(o - p_i) \cdot l_i < (p_i - k) \cdot l_i$ **then**
    // if obstacle is on the right hand side of the camera
    if $(o - p_i) \cdot r_i > 0$ **then**
      $u = -r_i v_{max}$ *// compute a desired velocity to the left*
    else
      $u = r_i v_{max}$ *// compute a desired velocity to the right*
    end if
    // subtract the current velocity to the desired force
    $F_{obs} = F_{obs} + (u - v_i)$
  end if
end for

Camera separation force:

moves a camera agent away from other camera agents that are too close and are looking in the same direction. In scouting mode, this force ensures a degree of diversity by separating similar cameras and locally improving the coverage of different events. In tracking mode, this same force enables the cameras to pick different views of the same event (useful when tracking a large group). The corresponding force is computed as described in Algorithm 4: if the distance between a camera agent c and a camera agent $i \neq c$ is lower than a threshold d_{max}, a force is applied either towards the right or the left of the camera. The intensity of this force is proportional to the angle between the look at vector l_i of camera i and the look at vector l_c of camera c, computed with $max(0, l_c \cdot l_i)$.

Algorithm 4 Camera separation: computes the sum F_{sep} of forces that pushes the camera c away from other camera agents. p_i and p_c represent the positions of camera agents i and c and r_c is the right vector of camera c.

for each camera i different from camera c **do**
  if $|p_i - p_c| < d_{max}$ **then**
    // move the camera to the left or to the right
    if $(p_i - p_c) \cdot r_c > 0$ **then**
      $u = -v_{max} r_c$ *// compute a desired velocity to the left*
    else
      $u = v_{max} r_c$ *// compute a desired velocity to the right*
    end if
    // subtract the current velocity to the desired velocity
    // and scale it
    $F_{sep} = F_{sep} + max(0, l_c \cdot l_i)(u - v_c)$
  end if
end for

Wandering force:

moves the camera in the scene while searching for new events in scouting mode. This force is computed by updating a wandering direction at each time step and computing a desired velocity from this direction (see [Rey99] for more details).

Containing force:

prevents the camera from wandering away in the scene. When the camera agent wander too far from the crowd, a steering force will push it back toward the center of the crowd (see [Rey99] for more details)

3.3.4 Camera steering torques

For the camera rotation, three torques were designed: aiming, camera avoidance and wandering.

Aiming torque:

rotates the camera towards the "optimal" orientation. This torque is inspired by Reynolds' arrival force (see Figure 3.2) and transposed to the problem of camera orientation. Given a desired camera orientation q_d, and the orientation q_i of a camera agent i, we need to compute the appropriate torque to reach the target camera orientation within the limits of a maximal rotational speed α_{max}. We first compute the difference between quaternions q_d and q_i to extract the angle q_α and axis q_a of rotation.

As defined in the arrival behavior (see Figure 3.2), if the angle α is below a given threshold α_t (meaning the angle is getting close to its desired value q_d), we progressively reduce the rotational speed. If the angle α is above the threshold, the rotational speed is set to a maximum value α_{max} at each time step. Algorithm 5 presents the computation of the torque T_{aim} depending on the angle α and the threshold angle α_t.

Algorithm 5 Aiming torque: computes a torque T_{aim} applied to camera c. The rotational velocity of camera c is denoted by $\dot{q}_c$.

// computing the quaternion difference between q_d and q_c
$q = q_d q_c^{-1}$
// q^α is the rotation angle of quaternion q
if $q^\alpha > \alpha_t$ **then**
  $\alpha = \alpha_{max}$
else
  $\alpha = \alpha_{max} q^\alpha / \alpha_t$
end if
// compute a rotation of angle α and axis q^a
$T_{aim} = (\alpha, q^a)\dot{q}_c^{-1}$

Figure 3.7 illustrates how the magnitude of the force (*i.e.* the value of the angle) is computed.

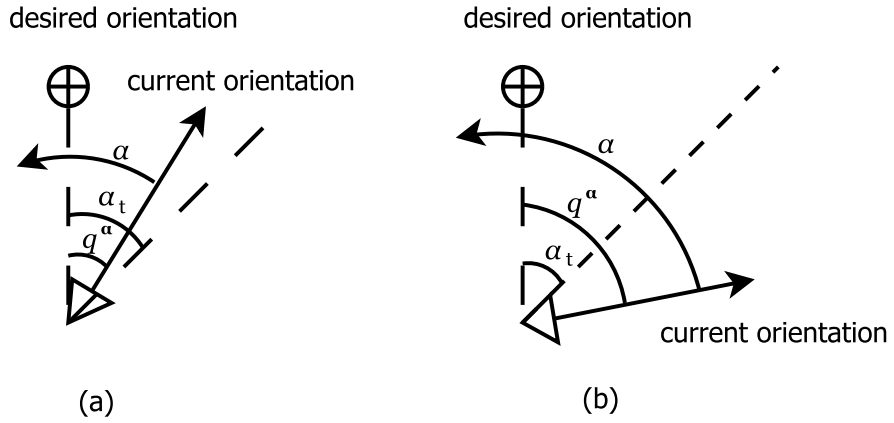


Figure 3.7: *Aiming torque inspired from the arrival behavior: two cases (a) q^α is less than α_t and (b) q^α is greater or equal to α_t.*

Camera avoidance torque:

turns cameras away from each other when they share part of their frustum.

The computation of this torque is two-fold. For each pair of cameras, we first check whether their frustums intersect. This collision test is simplified by using a 2D representation of the frustum. The intersection of the left and right axis (representing the frustum side planes) of the cameras are computed. Then the collision is detected if, for at least one of these four intersection points, their projections on the the forward vectors of the two cameras are contained inside the frustum of the respective cameras (i.e. the distance from the projection point to the camera origin is greater than the near plane distance of the frustum, and smaller than the far plane distance). Figure 3.8(a) shows an example where no collision are detected: the two projections P_j of each intersection point I_i on the forward vectors of the cameras are never inside both of the frustums. Conversely, Figure 3.8(b) shows an example of frustum intersection: the two projections $P1$ and $P2$ of the intersection point $I1$ are respectively inside the frustum of the camera 1 and 2. If a frustum collision is detected, the desired rotation velocity ($\dot{q}_d$) will

be oriented depending on the relative positions of the two cameras. The process is detailed in Algorithm 6.

Algorithm 6 Camera avoidance torque: computes a torque T_{avoid} for a camera c. The rotational velocity of camera c is denoted $\dot{q_c}$ and the right vector of camera c is denoted r_c. In addition, α_{vmax} represents the maximum rotational speed and u_c the up vector of the camera.

for all cameras i different from current camera c **do**
  if frustum of i intersects frustum of c **then**
    if $(p_i - p_c) \cdot r_c > 0$ **then**
      $\dot{q_d} = (-\alpha_{vmax}, u_c)$
    else
      $\dot{q_d} = (\alpha_{vmax}, u_c)$
    end if
    $T_{avoid} = T_{avoid}\dot{q_d}\dot{q_c}^{-1}$
  end if
end for

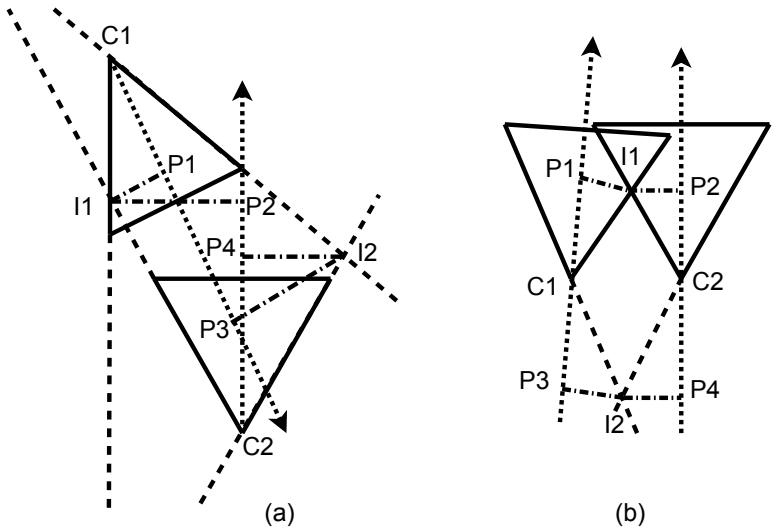


Figure 3.8: *Frustum collision test; the continuous lines indicate the frustum and the dotted arrows indicate the forward vectors.*

Camera wandering torque:

used for scouting new events. This torque is created by computing a wandering direction and aiming at a point along this direction.

3.4 EXPERIMENTAL RESULTS

For testing our approach, we implemented a realistic crowd simulation framework, based on a large number of characters (over a hundred) belonging to different ethnic groups. The task of

the camera agents is to report as many events as possible, and to record or broadcast the best possible coverage of those events. A video demonstrating the results is accessible online[1].

3.4.1 Crowd simulation

Our crowd simulation is actually based on Reynolds steering behaviors (see [LCG*13]). The scene simulates two distinct types of ethnic groups in Weld Quay, Malaysia back in the 19th century. The interactions of these two ethnic groups are highly influenced by their standard roles in the trading port. The main roles of each of these ethnic groups are as follows:

- Malay: local inhabitant/ seller at the market place
- Indian: imported worker

There are various interactions either between individual agents and within the same ethnic groups or among other different ethnic groups that are transpired in the trading port and we experiment our camera steering behaviors based on two scenarios as follows:

- Residents: Initialized interactions with any agents from the other ethnic group to sell their local products. These events are the one-to-one interactions that we are interested to identify.
- Workers: Unload the goods at the pier and download the goods at the containers. Emergents group movement should result from this shared goal.

The simulation takes place in a moderately complex environment (Figure 3.9) with the following scenario. At initialization, the workers are wandering in the market, the residents are trying to sell their products. When the boat arrives, some of the workers will try to reach the pier to unload the goods. When the boat is empty, it sails away and the workers can go back to the market place.

Figure 3.9: *Virtual environment of Weld Quay simulating the interactions between two ethnic groups in Malaysia back in the 19th century.*

[1] https://team.inria.fr/imagine/steering-behaviors/

3.4.2 Implementation details

For the purpose of efficiency, we used a quad-tree to accelerate crowd simulation and event detection. As a result, each camera agent is aware of all events and characters within its field of view. Figure 3.10 shows the 2D spatial layout of characters (red and blue dots) in an overview of our virtual environment.

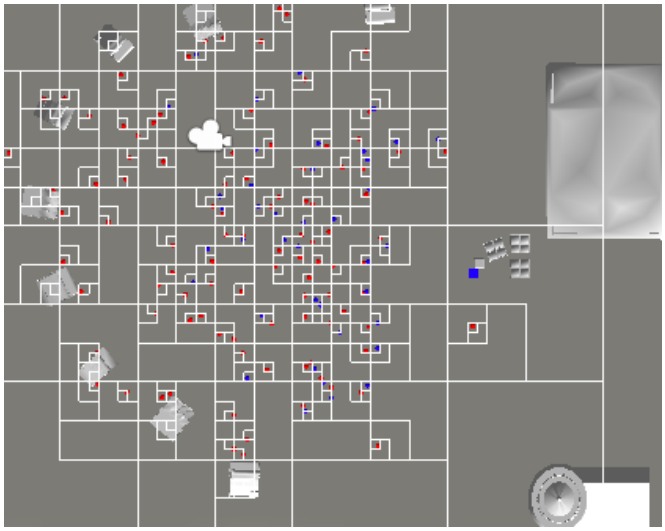


Figure 3.10: *Using a quad-tree representation to reduce computational cost in querying characters in camera frustums and detecting events.*

3.4.3 Qualitative evaluation

To evaluate our approach, we propose an *activity* metric that shows the overall presence of *active* characters (characters involved in an event) compared to *inactive* characters (characters not involved in an event). The activity metric is composed of a score representing the active characters and a score representing the inactive characters. These scores are evaluated for each camera by integrating a weight on characters: the closer a character is to the camera, the more he will affect the overall score of the camera.

Algorithm 7 Activity metric

```
for all cameras do
  for all characters seen by the camera do
    if character is Active then
      scoreActive += 1/distance(camera, character)
    else
      scoreIdle += 1/distance(camera, character)
    end if
  end for
end for
```

We compared the activity metrics for our system with a much simpler solution with static cameras placed at strategic positions (to ensure that they would not lose sight of the crowd –

information that our autonomous camera agents do not have). Figure 3.11(a) shows the temporal evolution of the ratio of active *vs.* idle characters captured by autonomous cameras. Figure 3.11(b) shows the results for static cameras (dark gray values correspond to active characters). We can see that, even though it is less stable (due to the scouting state) the average proportions of active characters remains more important than with statics cameras. Indeed, autonomous cameras try to get closer to the active characters, and thus improve scores whereas static camera are only waiting for active characters to pass by. We can observe the same phenomenon with eight static and moving cameras in Figure 3.12. Moreover, autonomous cameras are able to maintain a good framing of their targets while performing elaborate and dramatic camera motion, as can be seen in the accompanying video. Figures 3.3(a) and 3.3(b) display typical shots for one-to-one interactions and group motion events.

In Figures 3.11 and 3.12 we can observe some fluctuations in the graph. These fluctuations are due to the fact that when an event ends, it might not have any other events in its field of view since the camera was focusing on this specific event, and thus it sometimes results in a drastic drop in the ratio idle/active while the camera is searching for a new event.

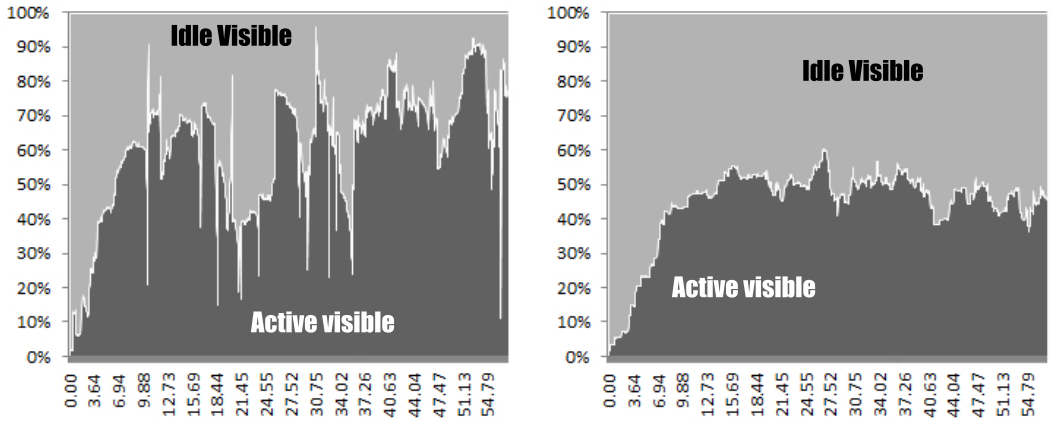
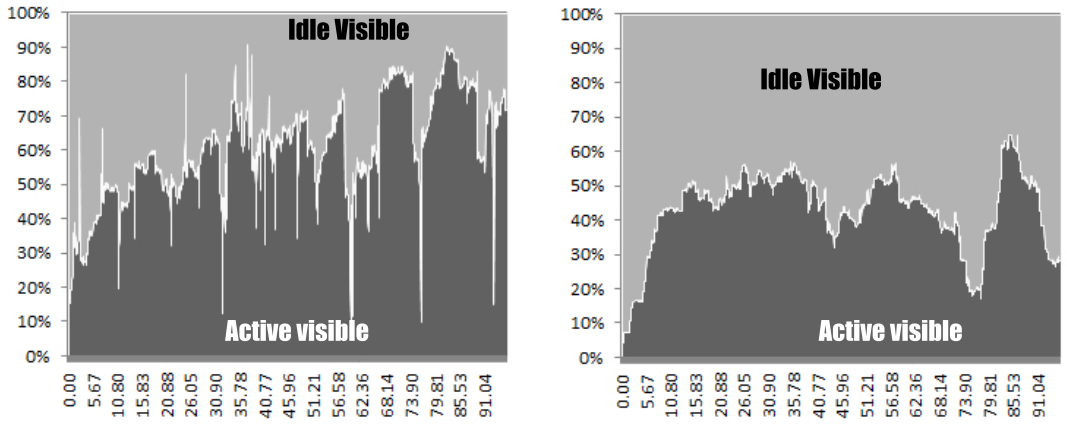


Figure 3.11: *Ratio between the active score and the idle score for groups of 4 autonomous (a) and static (b) cameras*

3.4.4 Quantitative evaluation

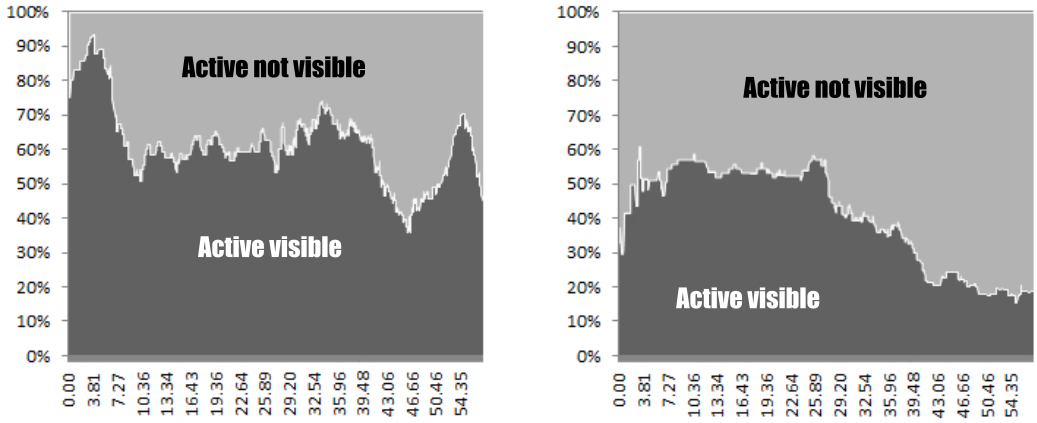
One way to present quantitative results is to express the number of active characters that are being viewed by the cameras, in comparison with the number of active characters not viewed by the camera. Results are reported in Figure 3.13 with 4 autonomous cameras and Figure 3.14 with 8 autonomous cameras. These results illustrate both the capacity of a small number of cameras to cover a crowd simulation (in comparison to 4 static cameras), and show that an increase in the number of cameras improves the results essentially for autonomous cameras.

In terms of performance, the method remains efficient and can steer 30 camera agents in real-time (15fps) on a Core i7@2.4GHz running Unity 4 with 100 virtual characters (see Figure 3.15). The bottleneck is essentially due to the number of virtual characters to simulate. The drop in framerate comes from the cross-computations between all camera agents necessary in the camera avoidance and camera separation forces.



(a) autonomous camera (b) static camera

Figure 3.12: *Ratio between the active score and the idle score for groups of 8 autonomous (a) and static (b) cameras*



(a) autonomous camera (b) static camera

Figure 3.13: *Temporal evolution of the ratio between active characters viewed by the cameras and active characters not viewed by the camera, considering 4 autonomous camera agents (a) and 4 static cameras (b).*

3.5 LIMITATIONS AND FUTURE WORK

At this stage, our model suffers from several limitations. The system was built to work with exterior environments and only handles two event classes (one-to-one interactions and group motions) which calls for the need to investigate a richer model of event categories [LCG*13, RS14]. Morever, the implementation only offers a single tracking mode. Other camera behaviors could easily be added as specialized idioms, in the fashion of [HCS96]. The focus in this chapter has been on the single task of tracking as many events as possible ; our solution could be extended to the case of coordinated cameras, where camera behaviors can be chosen by a "director" agent, taking higher level goals into account. Chapter 5 tackles several

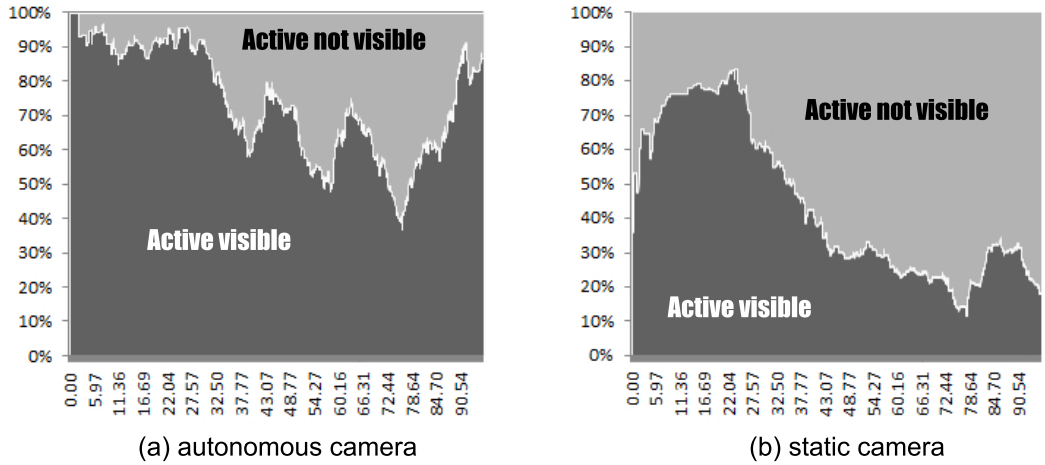


Figure 3.14: *Temporal evolution of the ratio between active characters viewed by the cameras and active characters not viewed by the camera, considering 8 autonomous camera agents (a) and 8 static cameras (b).*

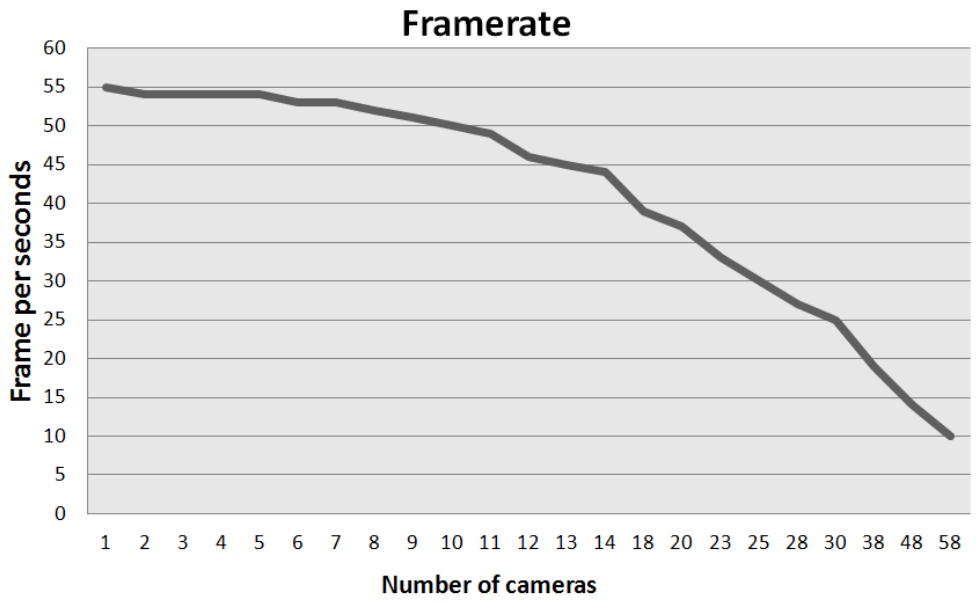


Figure 3.15: *Performance on crowd simulation when increasing the number of camera agents.*

of these limitations to address the challenge that is the computation of cinematic replays of game sessions.

Finally, this chapter was only dedicated to the task of controlling the cameras, without any consideration for editing. The system was designed to generate multiple camera shots and let an expert user deal with the editing. In the following chapter, we address this challenge and focus on the automation of the editing process to provide a tool that encompasses expert knowledge of editing principles [Ron12].

3.6 SUMMARY

In this chapter we have presented a novel approach to camera control through steering behaviors. The proposed method is used for the automatic computation of shots in crowd simulations using a predefined number of autonomous cameras. Experimental results show that the method provides a good coverage of events in moderately complex crowd simulations, with consistently correct image composition and event visibility. The autonomous cameras react to crowd animation events using specialized camera steering behaviors and forces based on Reynolds' model. By separately designing forces applied to the camera position, and torques applied to the camera orientation, our method offers a fine control over the camera agents. Overall, the strength of this approach lies in its simplicity and its ability to be extended with new steering behaviors.

CHAPTER

4

SEMI-MARKOV MODEL OF FILM EDITING AND APPLICATIONS

While the previous chapter focused on the problem of placing cameras to produce nice-looking views of the action, the problem of cutting shots from all available cameras was not addressed. In this chapter, we describe an optimization-based approach for automatically creating well-edited movies from a 3D animation.

After reviewing the main causes of editing errors in literature, we propose an editing model relying on a minimization of such errors. We make a plausible semi-Markov assumption, resulting in a dynamic programming solution which is computationally efficient. We also show that our method can generate movies with different editing rhythms and validate the results through a user study and a comparative analysis. Though this chapter is illustrated with manually placed cameras, this work is also used in chapters 5 and 6 with automatically computed shots.

4.1 INTRODUCTION

The wide availability of high-resolution 3D models and the facility to create new geometrical and animated contents, using low-cost input devices, open to many the possibility of becoming digital 3D storytellers. In chapter 3, we have described a new method for automating the positiong of the cameras. To date there is however a clear lack of accessible tools helping to easily perform the editing of such stories (selecting appropriate cuts between the shots created by the cameras). As seen in section 2.1.4, editing a movie requires the knowledge of a significant amount of empirical rules and established conventions. In particular *continuity editing* is a complex endeavor. Most 3D animation packages lack continuity editing tools, calling the need for automatic approaches that would, at least partially, support users in their creative process [DZO*13].

Previous contributions in automatic film editing have focused on generative methods mixing artificial intelligence and computer graphics techniques. However, evaluating the quality of film editing (whether generated by machines or by artists) is a notoriously difficult problem [LRGG14]. Some contributions mention heuristics for choosing between multiple editing solutions without further details [CAH*96] while other minimize a cost function which is insufficiently described to be reproduced [ER07]. Furthermore, the precise timing of cuts has not been addressed, nor the problem of controling the rhythm of cutting (number of shots per minute) and its role in establishing film tempo [ADV02]. As a result, most approaches tend to reproduce a reactive style of editing comparable to the *dragnet* style [Mur86], which mechanically cuts to new speakers or actions.

In this chapter, we propose a continuity editing model for 3D animations that provides a general solution to the automated creation of cinematographic sequences. Our model encodes the continuity editing process as a search for the optimal path through an *editing graph*. In this *editing graph*, a node represents a time-step (a temporal fragment of a shot), and an arc represents a transition between two cameras, going from a camera to either the same camera (no cut) or another camera (cut).

Our optimization uses dynamic programming to minimize, under a semi-Markovian hypothesis, the errors made along three criteria (see Figure 4.1): the quality of the shots (with respect to the unfolding actions), the respect of continuity editing rules and the respect of a well-founded model of rhythm (cutting pace). Semi-Markov models [Mur02, Yu10] have been used before in the context of information extraction [SC04], speech generation [ZTM*07] and computer vision [SWCS08]. To the best of our knowledge, this is the first time they are suggested as a computational model for film editing.

The contributions detailed in this chapter are: (i) a detailed formalization of continuity editing for 3D animation, encompassing a thorough number of visual properties and continuity rules (ii) an optimal algorithm for automatic editing in which parameters such as pacing can be controlled, thereby significantly increasing the expressiveness of editing tools, and (iii) a validation of our model through a user evaluation comparing the original edit of an existing movie with our optimal edit and with degraded approaches.

4.2 MOVIES AS SEMI-MARKOV CHAINS

In this chapter, we cast the problem of film editing as an optimization problem over a space of semi-Markov chains. Our system takes as input a 3D animation scene, comprising a flow of world events, and a set of rushes taken from different cameras and covering the whole scene. We then rank possible edits on three key aspects: (i) how much shots convey unfolding actions,

Figure 4.1: *Editing errors in a short sequence of a movie. From top to bottom: breaking the continuity errors; non-motivated shots and cuts. The last edit is the output of our system.*

(ii) how much continuity editing rules are enforced and (iii) how much an input cutting rhythm is respected.

Given a 3D animated scene with arbitrarily complex actions a and a choice of rushes (*i.e.* unedited footage) from M cameras, a semi-Markov chain is a sequence of states (shots) s_j with durations d_j, chosen according to a probability model over s_j, d_j and a. The probability of the next shot s_{j+1} of duration d_{j+1} and starting at time t_{j+1} depends only on the previous shot s_j and the actions a in the segment $[t_{j+1}, t_{j+1} + d_{j+1}]$.

We here introduce the notion of *editing graph*, the graph of all possible shots and transitions. In this graph, a node represents a time-step (one frame) of a rush and an arc represents a transition from frame i of a rush (camera) to frame $i + 1$ of a rush (same or different). The output of our system is then a full edit decision list (EDL) of the scene [AP12], which is computed as the continuous path through our editing graph minimizing the errors on these three key aspects.

The first input of our system is a symbolic representation of the story being represented in the virtual world. For convenience and simplicity, we summarize it with an ordered list of durative events represented expressed in the discrete event calculus [Mue07, Mue09]. The discrete event calculus is an appropriate representation for events taking place at discrete time steps (animation frames in our case), making it possible to reason about the consequences of events at intermediates time steps. We use it as a short-hand notation of events for all practical purposes.

Further, we extend the discrete event calculus for durative actions with the short-hand notation

$$HappensAt(t_1, t_2, Action(subject, verb, object))$$

with the meaning that a durative action takes place between times t_1 and t_2. This translates to two separate discrete event calculus formulas

- $HappensAt(t_1, StartAction(subject, verb, object)$
- $HappensAt(t_2, StopAction(subject, verb, object)$

with the result that $HoldsAt(t, Action(subject, verb, object)$ is true at all times $t \in (t_1, t_2)$.

In the following, we consider that the $subject$ and $object$ of all actions are characters, and we refer to the set of all characters as $\mathcal{C}$ and the set of all actions holding at a given time t as $\mathcal{A}(t)$. We use four main categories of actions: *speaking* actions performed by the character's mouth, *reacting* actions performed by the character's eyes, *manipulating* actions performed by the character's hands, and *moving* actions performed by the character's feet. As a result, a character can be the subject of at most four different actions at any given time and the object of an unlimited number of actions. Actions in our systems are typically generated by the animation system, but only a subset of the animation is labeled as action.

The second input of our system is a list of M rushes from different cameras filming the scene for a total duration of N video frames. We are agnostic about how the rushes are obtained.

The output of our system is a movie, described as an edit decision list (EDL) defined as a sequence of shots s_j in the form of triplets (r_j, t_j, d_j). Note that in this chapter, we only consider chronological EDLs where time is preserved ($t_{j+1} = t_j + d_j$). In this limited context, the EDL can be reconstituted using the rush selection function $r(t)$ which gives the rush index as a function of time.

We here propose to cast the editing process into a mathematical model accounting for three criteria: (i) how much shots convey unfolding actions, (ii) the continuity editing principles and (iii) the cutting rhythm of the movie. To do so, we use a log-linear model where the probability of choosing a particular sequence of shots s_j is taken to be the exponential of a linear cost function $C(s, a)$. The cost function $C(s, a)$ is further decomposed into three terms which separately measure (i) errors in conveying unfolding actions in each shot, (ii) violations of continuity editing rules in each cut and (iii) errors in choosing shot durations.

$$C(s,a) = \sum_j \sum_{t_j \le t \le t_j + d_j} C^S(r_j, t) + \sum_{1 \le j} C^T(r_{j-1}, r_j, t_j) + \sum_j C^R(d_j) \tag{4.1}$$

In this equation, the first term is a sum over all frames of all shots of a cost function C^S related to shot quality (see section 4.3). The second term is a sum over all cuts of a cost function C^T related to transitions between shots (see section 4.4). Those two term are further decomposed into weighted sums of features, *i.e.* $C^S(r_j, t) = \sum_k w_k^S C_k^S(r_j, t)$ and $C^T(r_i, r_j, t) = \sum_k w_k^T C_k^T(r_i, r_j, t)$. The third term is a sum over all shots of a cost function C^R related to editing rhythm (see section 4.5). In the following sections, we explain in details each of those terms.

4.3 Measuring Shot Quality

In video editing, shot selection is performed based on two different aspects: the visual quality and the narrative relevance of the shots [TB98, TB93]. Professional film editors use a complex combination of aesthetic judgements, common sense reasoning, deep understanding of the story and the psychology of the targeted audience to make decisions. In addition, they use their personal experience and knowledge of film history. In this work, we avoid the difficult task of understanding what makes a "good shot'" or a "best shot" and instead use a measure of what certainly makes a "bad shot". With respect to narrative relevance, a shot that presents irrelevant information is almost certainly wrong. Thus, the relative sizes of actors and actions should remain consistent with the relative importance in the story. This general principle was phrased out by Alfred Hitchcock [TS67] and can be used to discard narratively irrelevant shot choices.

(a) *Speak* **(b)** *React* **(c)** *Manipulate* **(d)** *Move*

Figure 4.2: *Symbolic projection of all 4 action categories.*

Similarly, we evaluate the visual quality of a shot by counting the number of composition errors, based on established practices.

In this section, we first describe how the different elements of the environment (objects and characters) are identified in the rushes through a stage of *symbolic projection*. We then explain the computation of narrative importance, and using these two concepts, we detail the computation of the different cost functions that evaluate the narrative relevance of the shots. Finally we describe the cost functions used to evaluate the quality of the framing and ensure the visual aesthetic of the final edit.

4.3.1 Symbolic projection

During the shooting of a virtual scene, all cameras capture images which are perspective projections of the world scene into their own frame of reference. In parallel to the computation performed by the graphics pipeline, we perform a symbolic projection of the actions which keeps a record of *how much of the action is visible* in each rush at every single frame.

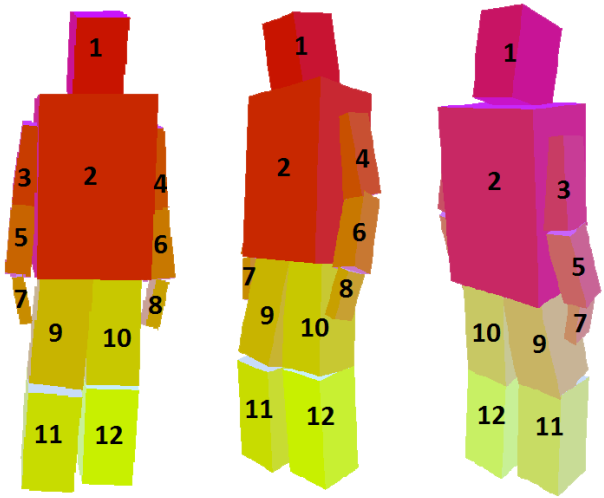
This is performed as follows. First, each action is decomposed into its constituents – verb, subject and object. Based on the verb category (speaking, reacting, moving or manipulating), we then compute the bounding boxes of involved body parts (see Figure 4.3) of the subject and object characters. We then compute the screen size of their projection in each frame of a rush (see Figure 4.2).

Second, to evaluate how much of these actions are visible, we compute the visible and occluded areas of characters. To do so, for each face f of each body part b of a character, we compute its projected size (or area) $S(f,r,t)$ at time t in rush r. This projected size is measured relatively to the screen size, and comprises both the on-screen and off-screen projections of f. We then define the visible and occluded sizes of f as follows. Its occluded size $O(f,r,t)$ corresponds to the cumulative size of its areas that are either occluded or appear off-screen, while its visible size $V(f,r,t)$ is the complementary value computed such that $S(f,r,t) = V(f,r,t) + O(f,r,t)$. We finally define the projected size and the visible size of each character c as the sum of corresponding values on each face of its body parts:

$$V(c,r,t) = \sum_{b \in c} \sum_{f \in b} V(f,r,t)$$

$$S(c,r,t) = \sum_{b \in c} \sum_{f \in b} S(f,r,t)$$

This method is further easily extended to the case of non-character objects (we use their bounding boxes) and multiple-characters.



ID	Name	Binary	Hexa
1	Head	000000000001	0x001
2	Chest	000000000010	0x002
3	Right upper arm	000000000100	0x004
4	Left upper arm	000000001000	0x008
5	Right forearm	000000010000	0x010
6	Left forearm	000000100000	0x020
7	Right hand	000001000000	0x040
8	Left hand	000010000000	0x080
9	Right Thigh	000100000000	0x100
10	Left Thigh	001000000000	0x200
11	Right Calf	010000000000	0x400
12	Left Calf	100000000000	0x800
3, 5, 7	Right arm	000001010100	0x054
4, 6, 8	Left arm	000010101000	0x0A8
1 ... 8	Upper body	000011111111	0x0FF
9 ... 12	Legs	111100000000	0xF00
1 ... 12	Full body	111111111111	0xFFF

Figure 4.3: *Body parts defined on each character.*

4.3.2 Narrative importance

We evaluate the narrative relevance of shots with a measure of the importance of the actions they present to the viewer. In an ideal world, that information would be inferred from the movie script, or annotated by the film director. In our system, we compute the narrative importance of actions based on a heuristic decomposition of the cast into main characters and secondary characters. We label all actions involving the main characters as *foreground* actions, and all other actions as *background* actions. To each action a holding at time t, a narrative importance $I(a)$ is attributed depending on whether a is a foreground or a background action. The total importance of actions at time t is then given by

$$I_{tot}(t) = \sum_{a \in \mathcal{A}(t)} I(a)$$

Each action type also defines a distribution of importance on the subject and object characters that sums to 1. For instance, in a speak action, the subject have an importance of 60%, while the object have an importance of 40%. The overall importance of a character c is then given by

$$I(c,t) = \sum_{a \in \mathcal{A}(t)} I(a) \cdot I(c|a)$$

where $I(c|a)$ represents the importance of character c in action a. Note that the same computation can be performed based on other heuristic rules on what constitutes a foreground or background action.

In the same way, actions define a distribution of importance on the body parts of subject and object characters which also sums to 1 on each character. For instance, in a speak action the subject's and object's visual targets are their heads (importance of 100%), in a manipulate action the subject's visual target is its head and chest (importance of 50% for each) and in a move action, the subject's visual target is its full body (the importance is equally distributed on each body part). The overall importance of a body part b belonging to character c is finally

(a) *Bad action visibility* **(b)** *Good action visibility*

Figure 4.4: *Examples of bad(left) and good(right) action visibility.*

given by

$$I(b,t) = \sum_{a \in \mathcal{A}(t)} I(a) \cdot I(c|a) \cdot I(b|a)$$

where $I(b|a)$ represents the importance of body part b in action a.

Finally the importance attributed to a body part is non-uniformly distributed on the different faces of its bounding box. We arbitrarily defined values based on commonsense estimations. For the head or chest, for instance, front is given an importance of 50%, sides is given an importance of 15%, top is given an importance of 10% and bottom/back are both given an importance of 5% each. Since, at any time, no more than 3 faces are visible, this importance is finally normalized through a division by the maximum value possible (front, one side and top: 75%). The overall importance of a face f belonging to body part b of character c is finally given by

$$I(f,t) = \sum_{a \in \mathcal{A}(t)} I(a) \cdot I(c|a) \cdot I(b|a) \cdot I(f|b)$$

where $I(f,t)$ is the importance of the face (front,back,top,bottom or side) on the body part at a given time t.

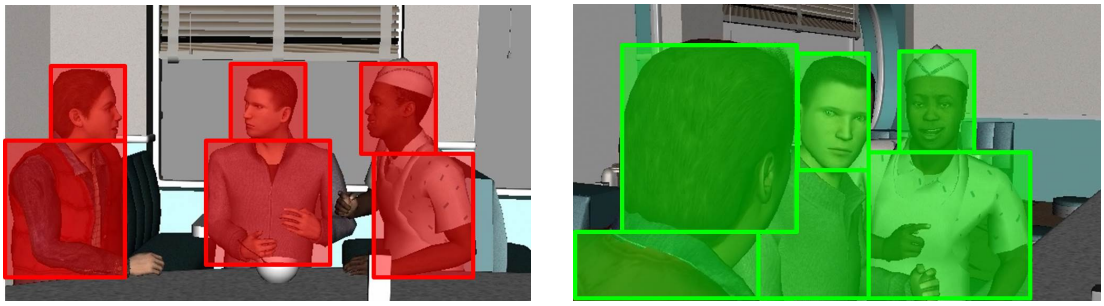
4.3.3 Narrative relevance

Based on the symbolic and geometric data related to unfolding actions of a given frame, we evaluate every frame of every shot on three aspects: the action visibility, the action proximity and the action ordering (also known as the Hitchcock principle).

The action visibility term evaluates how much of unfolding actions is visible (see Figure 4.4). To fully satisfy this criteria, each important body part of a character taking part in an unfolding action should be on-screen and fully visible. The cost associated to action visibility is computed as the sum, on each face f of each body part b of each character c, of the occluded proportion of the face weighted by its narrative importance:

$$C_V^S(r,t) = \sum_{c \in \mathcal{C}} \sum_{b \in c} \sum_{f \in b} I(f,t) \cdot \frac{O(f,r,t)}{S(f,r,t)}$$

The action proximity term evaluates how immersed the camera is in the unfolding actions, *i.e.* how much the screen is filled by actions (see Figure 4.5). The cost (or penalty) associated



(a) *Bad action proximity* **(b)** *Good action proximity*

Figure 4.5: *Examples of bad(left) and good(right) action proximity.*

to poor action proximity is then given by the proportion of the screen filled by the characters:

$$C_P^S(r,t) = 1 - \sum_c V(c,r,t)$$

The action ordering term evaluates how much the on-screen importance of a character matches its narrative importance. This term is most closely related with the original Hitchcock principle (which states that the size of a character should be proportional to its narrative importance in the story [TS67]). Previous work has proposed several partial implementations of the Hitchcock principle [Haw05, DeL09], which all have shortcomings. Our implementation considers all characters present in the scene, not just the characters present in each shot, or the characters participating in the main action. This has the benefit to easily rule out prominent shots of unimportant characters and favor prominent shots of important characters, focusing on the relevant body parts (mouth while speaking, eyes while reacting, hands while manipulating, feet while moving). The cost associated to the Hitchcock principle is computed as the sum of all deviations of the on-screen visibility of a character compared to its narrative importance:

$$C_H^S(r,t) = \sum_{c \in \mathcal{C}} \left| \frac{I(c,t)}{\sum_{c'} I(c',t)} - \frac{V(c,r,t)}{\sum_{c'} V(c',r,t)} \right|$$

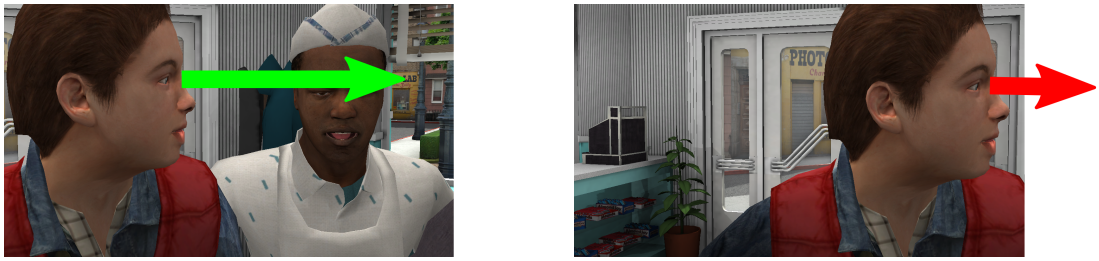
Figure 4.27 shows the correlation between the narrative importance of characters and their on-screen sizes. An important result of our work is that the three terms $C_V^S(r,t)$, $C_P^S(r,t)$ and $C_H^S(r,t)$ contribute to the narrative relevance of shots in complementary ways and must be taken into account together, rather than separately.

4.3.4 Visual quality

Previous cost functions focus on the narrative relevance of the shots regarding the story but do not account for any visual aesthetic. To consider visual quality, we analyse frame composition on two separate directions: horizontal and vertical. This approach does not intend to guarantee a perfect frame composition but rather to eliminate shots that break basic composition rules described in section 2.1.4.

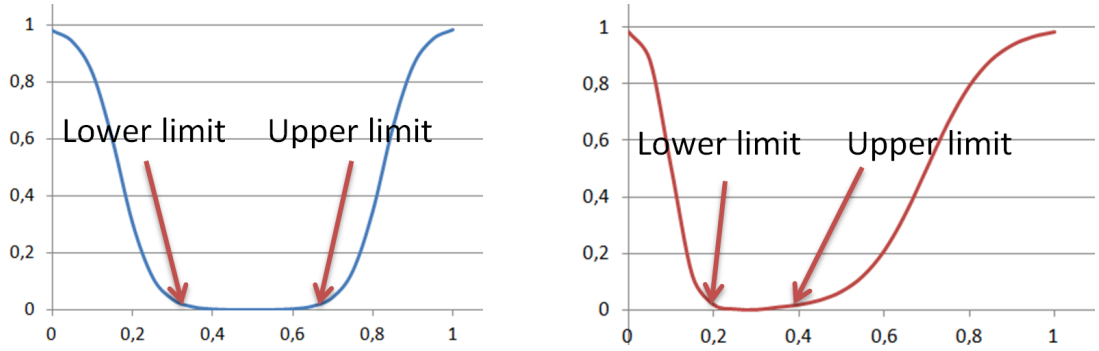
Horizontal composition

First, at each frame, we study the horizontal composition of the characters in each rush. Following common practices, we favor shot composition where each character is given enough screen



(a) *Good look-room* **(b)** *Bad look-room*

Figure 4.6: *Examples of good/bad horizontal composition.*



(a) ϕ_{HCmin} *when the character is aligned with the camera:* $\theta = 0^o$ **(b)** ϕ_{HCmax} *when the character is perpendicular to the camera:* $\theta = 90^o$

Figure 4.7: *Cost functions of the horizontal composition for an actor facing the camera (a) and perpendicular to the camera (b)*

space relative to the image frame, especially in the direction where he is looking (referred here as look-room). Figure 4.6 gives an example of bad and good horizontal frame composition.

The cost function for the horizontal composition is given by:

$$C^S_{HC}(r,t) = \sum_c v(c) \cdot \phi_{HC}\left(eyes^x_r, \theta\right)$$

where $\phi_{HC}(x,\theta)$ is a non-linear piecewise-defined function. It takes as input $eyes(c,r)_x$, the x coordinate of the eyes position in the screen space of character c in the rush r and θ, the angle between the camera and characters' orientation. ϕ_{HC} returns a maximum value when the eyes are at the extreme sides of the screen or when the character is near and facing a side of the screen. Figure 4.7 shows $\phi_{HCmin}(x)$ and $\phi_{HCmax}(x)$: the functions for the extreme values of θ. They both use sigmoids defined by different upper and lower limits. For any θ, ϕ_{HC} is computed by interpolating the lower and upper limits from these extreme cases.

Figure 4.8 demonstrates the importance of handling horizontal composition. A set of similar viewpoints is created by degrading the composition of a shot with small variations of its horizontal composition. Given this set of possibilities, our system successfully select the properly composed shot.

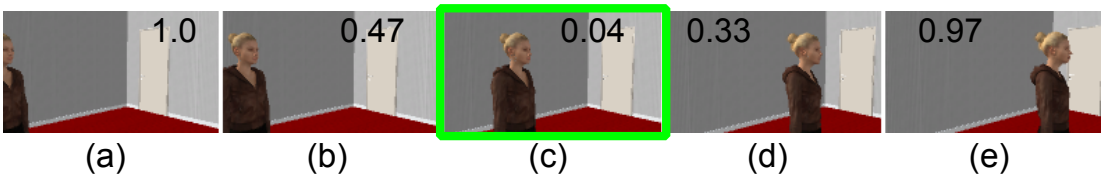


Figure 4.8: *Evaluation of different shot with regard to the horizontal composition. The computed cost is displayed for each shot and the selected shot is highlted in green.*

(a) *Good vertical composition* **(b)** *Bad vertical composition*

Figure 4.9: *Examples of good (left) and bad (right) vertical composition.*

Vertical composition

We then study, at each frame, the vertical composition of the characters for each rush. Following established practices in cinematography, we penalize poor vertical composition based on two distinct rules: rule of the third and headroom. The rule of thirds consist in placing the characters' eyes at the third of the screen. The headroom is a rule on the vertical framing of a character, and is commonly measured as the distance between the top of a character's head and the top of the screen. It is important to make the headroom not too much or too little. Figure 4.9 gives an example of bad vertical composition. The cost function penalizing poor vertical composition is defined as followed:

$$C^S_{VC}(r,t) = \sum_{c \in \mathcal{C}(t)} \left| \frac{2}{3} - eyes(c,r)_y \right| + \phi_H(top(c,r)_y)$$

where $eyes(c,r)_y$ is the y coordinate of the eyes position in the screen space of the rush r and ϕ_H is a non-linear function taking in parameter $top(c,r)_y$, the y coordinate of the top of the head. ϕ_H returns a maximum value when the top of the head is at the top of the screen and a minimum value when it is further than a threshold distance (10% of the height of the screen for example).

Figure 4.10 illustrates the importance and validity of this approach. A set of similar viewpoints is created by degrading the composition of a shot with small variations of its vertical composition. Given this set of possibilities, our system successfully select the properly composed shot.

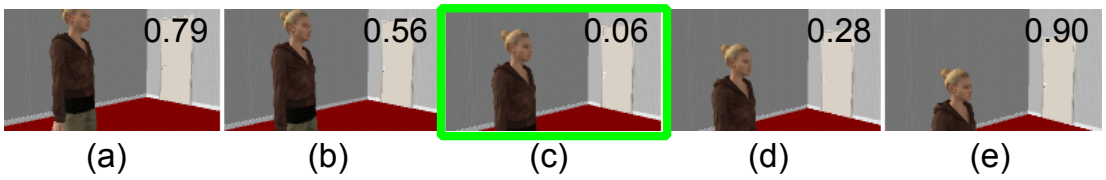


Figure 4.10

4.4 Measuring Cut Quality

In this section, we propose a solution to evaluate the quality of the cuts. More precisely, we use a computational model of continuity-editing to penalize poor cutting decisions. Continuity-editing is the most commonly used editing style in filmmaking. It relies on a set of well established rules: enforcing screen, motion and gaze continuity, maintaining the left-to-right ordering of on-screen characters, and avoiding jump cuts[Dmy84, TB98]. Jump cuts attract the attention of audience and create the illusion that the actors, rather than the camera, change positions during the cut. They are sometimes referred to as *first-order editing errors*, while screen, motion, gaze and left-to-right order continuity errors are considered *second–order editing errors* [dDR98]. Preserving continuity in the screen positions, motion directions and gaze directions of actors as well as their left-to-right ordering makes it easier for the audience to build a consistent three-dimensional representation of the scene [Smi05, Gd07, BC11]. Third-order editing errors are caused by violations of the causal order of actions in the scene, and are not directly addressed in this work. As a foundation for future work, we provide formulas for systematically detecting and quantifying first-order and second-order editing errors according to the elementary rules of continuity editing, given the symbolic projection of the scene in the two shots. Unlike previous work, our formulas are independent of the number of characters present in the two shots, and can therefore be used to evaluate the quality of the cut between shots of arbitrary complexity (number of actors, depth of field, shot sizes).

4.4.1 Screen continuity

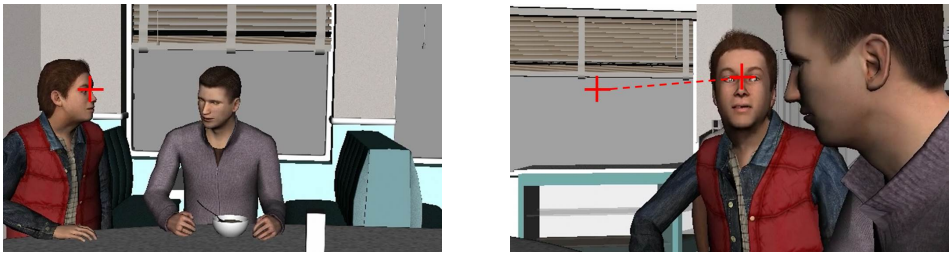
Spatial continuity is essential to ease the transition between two shots and enforce the visual fluidity of the cut. It prevents the disorientation of the audience.Figure 4.11 illustrates the problem of screen discontinuity. We penalize such a discontinuity by summing, on each single character, its screen position change (in screen space coordinates). The associated cost function is defined as follows

$$C_S^T(r_1, r_2, t) = \sum_c v(c) \cdot \phi_S \left(P(c, r_2) - P(c, r_1) \right)$$

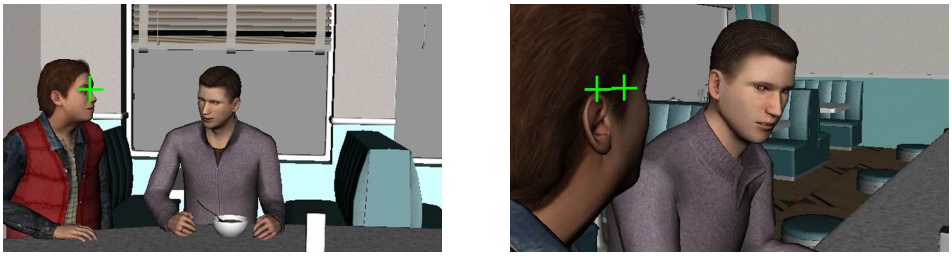
where $P(c, r_1)$ and $P(c, r_2)$ represent the 2D screen position of character c resp. before and after the cut. ϕ_S is a non-linear function which takes as input the distance between both positions. It then returns the minimum penalty (0) for two identical positions, and the penalty increases with their distance.

4.4.2 Motion continuity

Motion of the characters on the screen also affects the quality of a cut. Transitions between two shots where one or more characters have a different perceived motion (in screen space)



(a) *Screen discontinuity*



(b) *Screen continuity*

Figure 4.11: *Examples of cuts violating (top) or respecting (bottom) screen continuity.*

have a disorienting effect on the audience. This problem of motion continuity is illustrated in Figure 4.12. To penalize such discontinuity, a cost function C_M^T is computed by summing, on each single character, its change of apparent motion direction.

$$C_M^T(r_1, r_2, t) = \sum_c v(c) \cdot \phi_M \left(M(c, r_1), M(c, r_2)\right)$$

where $M(c, r_1)$ and $M(c, r_2)$ are 2D vectors representing the on-screen motion direction of character c resp. before and after the cut. ϕ_M is a non-linear function which takes as input these two consecutive motion directions. It then returns the minimum penalty (0) when the two vectors are close enough (*e.g.* a character moving in a given direction keeps moving in a similar direction after the cut), and the penalty increases as these vectors differ from each other.

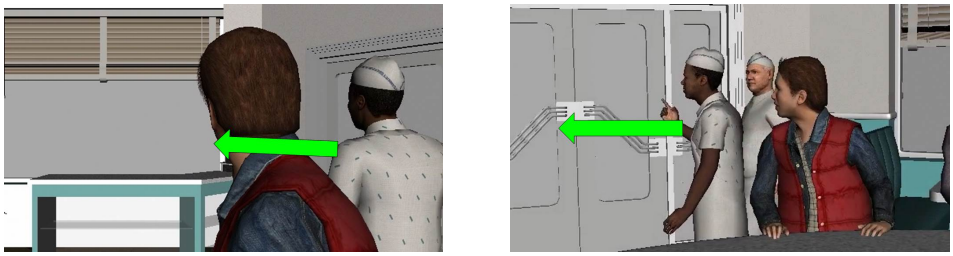
4.4.3 Gaze continuity

Visual discomfort may also happen in a transition between two shots when characters involved in both shots are looking in different direction on the screen before and after the cut. An illustration of gaze continuity is given in Figure 4.13. In a similar way to the motion continuity, we penalize such a discontinuity by summing, on each single character, its change of apparent gaze direction. The cost function is given

$$C_G^T(r_1, r_2, t) = \sum_c v(c) \cdot \phi_G \left(G(c, r_1), G(c, r_2)\right)$$

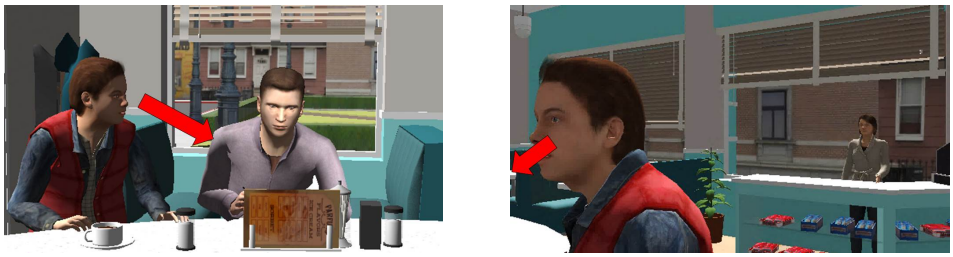
where $G(c, r_1)$ and $G(c, r_2)$ are 2D vectors representing the on-screen gaze direction of character c resp. before and after the cut. ϕ_G is a non-linear function which takes as input these two consecutive gaze directions. It then returns the minimum penalty (0) when the two vectors are close enough (*e.g.* a character looking in a given direction keeps looking in a similar direction after the cut), and the penalty increases as these vectors differ from each other.

(a) *Motion discontinuity*

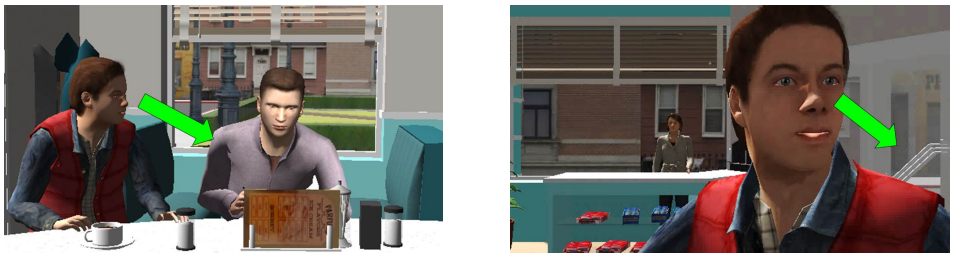


(b) *Motion continuity*

Figure 4.12: *Examples of cuts violating (top) or respecting (bottom) motion continuity.*



(a) *Gaze discontinuity*



(b) *Gaze continuity*

Figure 4.13: *Examples of cuts violating (top) or respecting (bottom) gaze continuity.*

4.4.4 Left-to-right ordering

The left-to-right ordering of characters is another important factor to enforce visual continuity. Characters whose relative screen positions are reversed after a cut appear to be jumping around, which attracts attention to the cut [Smi05] – this criteria is also known as the 180-degree rule. An illustration of the left-to-right continuity rule is given in Figure 4.14. We penalize such a discontinuity by summing, on each pair of characters (c, c'), their change in relative on-screen position (from left to right, this is also known as the 180-degree rule). To do so, we define

(a) *Left-to-right ordering discontinuity*

(b) *Left-to-right ordering continuity*

Figure 4.14: *Examples of cuts with inconsistent (top) and consistent (bottom) left-to-right ordering.*

a new weighting factor $v(c, c')$ computed as the product $v(c) \cdot v(c')$ of the weights of both characters. We then give no importance to a pair of characters where at least one is off-screen either before or after the cut, little importance to a pair of background characters, and much importance to a pair of foreground characters. Practically, this penalty is computed as follows

$$C_L^T(r_1, r_2, t) = \sum_{c,c'} v(c, c') \cdot \phi_L \left(L(c, c', r_1), L(c, c', r_2) \right)$$

where $L(c, c', r_1)$ and $L(c, c', r_2)$ are two real values representing the relative position of characters c and c' resp. before and after the cut (practically, this relative position is computed as the signed difference of their on-screen horizontal coordinates). ϕ_L is a non-linear function taking as input these two reals. It then returns the minimum penalty (0) when both values are of same sign (*i.e.* the relative position of characters is enforced) and the maximum penalty (1) when the two values are of opposite sign (*i.e.* the relative position of characters is reversed).

4.4.5 Jump cuts

In film editing, a jump cut is defined as a cut in which two sequential shots of the same subject are taken from similar camera positions. This type of edit gives the effect of jumping forwards in time. An illustration of jump cut is given in Figure 4.15. We penalize a jump cut by summing, on each single character, the degree of similarity (both in size and view angle) between the two frames before and after the cut. Practically, this penalty is computed as follows

$$C_J^T(r_1, r_2, t) = \sum_{c} v(c) \cdot \phi_J \left(\Delta S(c, r_1, r_2), \Delta\theta(c, r_1, r_2) \right)$$

where $\Delta S(c, r_1, r_2)$ and $\Delta\theta(c, r_1, r_2)$ are the differences in resp. apparent size and view angle of character c between the two frames. ϕ_J is a non-linear function taking these two parameters

(a) *Insufficient change in both size and view angle*

(b) *Change in apparent size*

(c) *Change in view angle*

Figure 4.15: *Example of a jump cut (top) where the main character is seen with similar shot sizes and view angles. Changing the shot size (middle) or the view angle (bottom) prevents the jump cut impression.*

as input. It considers a minimum acceptable change in apparent size ΔS_{min}, as well as a minimum acceptable change in view angle θ_{min} (often set to 30 degree). ϕ_J then returns the maximum penalty when no change occurs neither in apparent size nor view angle of character c, and the penalty decreases as the change in either apparent size or view angle increases.

4.5 Measuring Rhythm Quality

Cutting rhythm is an important element of film editing style [Bor98]. Cutting between cameras produces visual rhythm. Fast cutting as in an action scene can change cameras as often as every half second. The cutting rhythm has been studied extensively by film scholars [Sal09, CDN10, Sal11, DBC12] who have shown that it is well approximated with a log-normal distribution of shot durations, and that the mean and standard deviations (SD) of the log-normal distribution can be used to summarize the editing style of the movie. Recently, that claim was challenged by Redfern who showed that statistical tests in fact rejected the log-normal hypothesis in a majority of cases [Red15]. For our purpose, it is sufficient that the distribution of shot duration is qualitatively similar to a log-normal distribution. Furthermore, we have experimentally verified

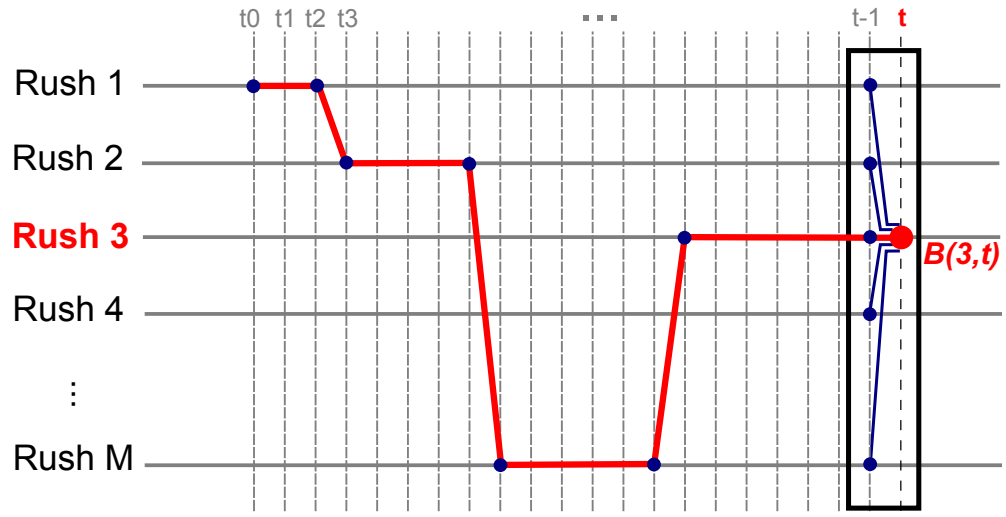


Figure 4.16: *Illustration of a dynamic programming algorithm, using Markov decisions. The best edit in which a shot is in Rush 3 at time t only depends on the best edits at time $t-1$ and the transitions from Rushes $r_0 \neq 3$ to Rush 3 between $t-1$ and t.*

that the log-normal hypothesis is well verified at the scale of a movie scene or sequence, even it may be rejected at the scale of an entire movie. Based on those findings, we propose to measure the rhythm quality of an edited sequence by counting the number of outliers to a given log-normal distribution.

Parameters of the log-normal distribution are the mean μ and standard deviation σ of the log-transformed durations $\log d_j$, which result in a skewed distribution of durations with average shot length $ASL = \exp\left(\mu + \frac{\sigma}{2}\right)$ and variance $Var = \exp\left(2\mu + \sigma^2\right)\left(\exp\sigma^2 - 1\right)$ [LSA01]. Rather than making automatic decisions, our system is designed to let the user/director choose the average shot length (ASL) which dictates the rhythm of editing, and hence the editing style. To enforce those values, we compute a cost measuring, for each shot s_j of duration d_j, the deviation of its duration from the log-normal distribution

$$C^R(d_j) = \frac{(\log d_j - \mu)^2}{2\sigma^2} + \log d_j$$

4.6 Optimizing over Semi-Markov Chains

We evaluate the cost of an arbitrary edit decision list of shots s_j with a weighted sum of simple feature functions. To do so, we use a dynamic programming approach to find the minimum cost solution by storing partial solutions [MHJ95, Mur02].

Figure 4.16 and Figure 4.17 illustrate two different approaches to the problem. In the first approach (see Figure 4.16), we consider a classical Markov model. The Markov chain is a sequence of states (cameras) s_j where the probability of the next state s_{j+1} depends only on the previous state s_j and the actions a at time t_{j+1}. This approach is limited as it does not account for the rhythm of the sequence (given by the duration of the shots) and thus, does not explore every solutions. Using a semi-Markov, model our approach (see Figure 4.17) better suits the problem. The semi-Markov chain is defined by a sequence of states s_j with durations d_j, where the probability of the next state s_{j+1} and duration d_{j+1} starting at time t_{j+1} depends

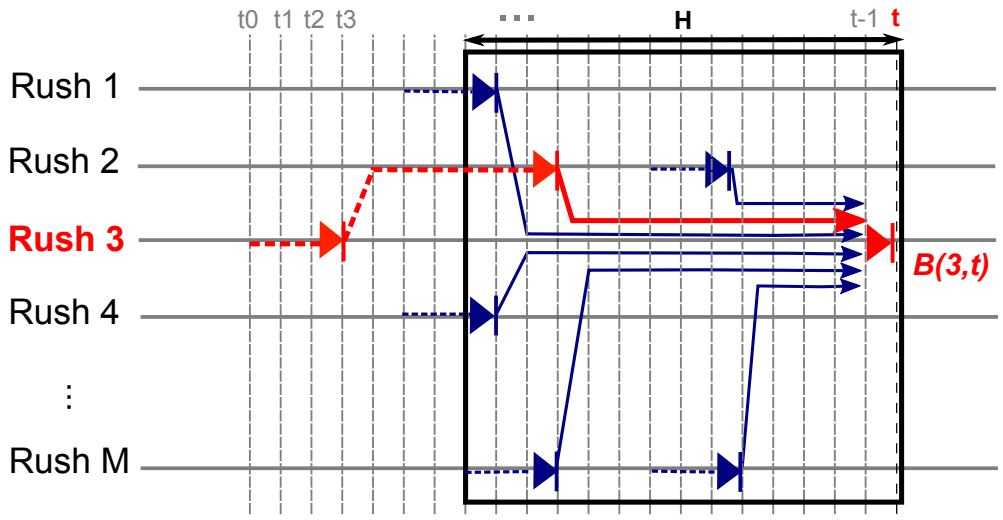


Figure 4.17: *Illustration of our dynamic programming algorithm, using semi-Markov decisions. The best edit in which a shot ends in Rush 3 at time t is computed as the best combination (drawn in red) of an edit in which a shot ends in Rush $r_0 \neq 3$ at a prior time $t_i < t$ then a shot using Rush 3 between t_i and t.*

only on the previous state s_j and the actions a in the segment $[t_{j+1}, t_{j+1} + d_{j+1}]$. This model retains the benefits of the dynamic programming approach while significantly increasing its expressiveness.

We define $B(r,t)$ to be the cost of the best sequence of shots ending at time t with a shot using rush r. One important result that follows from our choice of cost functions is the following recurrence relation

$$B(r,t) = \min_{\substack{t_0 < t \\ r_0 \neq r}} \left[B(r_0,t_0) + C^T(r_0,r,t_0+1) + \sum_{t'=t_0+1}^{t} C^S(r,t') + C^R(t-t_0) \right] \quad (4.2)$$

In plain words, the best sequence ending on rush r at time (frame) t can be computed by comparing all combinations of a shot ending on rush r_0 at time $t_0 < t$, followed by a cut from rush r_0 to rush r between frames t_0 and $t_0 + 1$, followed by a shot of duration $t - t_0$ using rush r (see Figure 4.17).

The weights of the feature functions were first arbitrarily defined based on simple assumptions. For the shot selection, we gave the Hitchcock principle and the action visibility the highest weights, as the main objective of editing remains the communication of narrative information – sometimes at the expense of visual quality. For the cutting criteria, we stressed the importance of the rules which violations are the most disturbing to the audience, namely the jump cuts and 180-degree rules. We then adjusted all these parameters with the dataset detailed in section 4.7.1. For instance, we increased the weight for the action proximity in order to favor close shots (such as over the shoulder shots) to more distant ones (such as apex shots). With our system, we provide a default set of parameters that can be adjusted to suit other cinematographic styles. The pacing especially should be adapted to match the desired rhythm. Finally, we tested this set of values with various camera inputs in different environments (see sections 5.6,6.5.3 and 6.5.4).

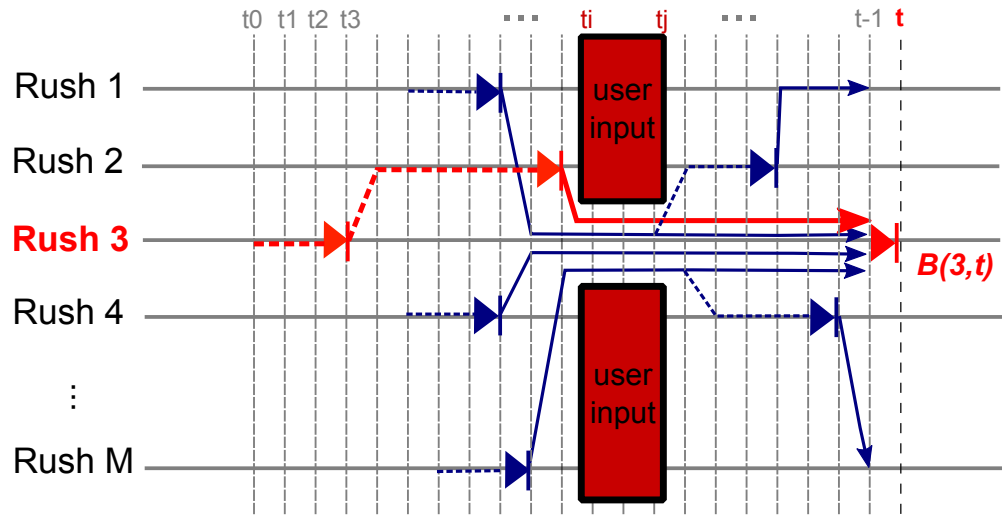


Figure 4.18: *Illustration of film graph pruning. Every possible edit necessarily uses rush* 3 *between* t_i *and* t_j *as specified by the user. Any other rush is removed from the graph between* t_i *and* t_j.

This optimization-based approach allows to automatically compute edits without the need for interaction with an expert user. However, if a user wishes to contribute to the edit, our solution allows to enforce the choice of camera at any moment in time. To account for such constraints in the computation of the optimal edit, the system performs a straightforward pruning on the film graph as illustrated in Figure 4.18. User-constraints of this type can be useful for enforcing higher-level stylistic effects that would be otherwise difficult to obtain with a purely automatic method, i.e. book-ending (forcing the sequence to begin and end on the same shot). The user can separately specify the starting and ending shots as user-constraints, resulting in a book-ending structure, without sacrificing the efficiency of the method.

4.7 Experimental results and validation

The evaluation of editing is a general and challenging problem [LRGG14] since no ground truth is available for objective comparisons. As a consequence, the quality of an edit can only be measured subjectively through indirect user evaluations.

As a result, we validated our approach with a subjective user study comparing our solution with various baseline methods using the same set of rushes. This section describes our data set and experiments, and discusses their results quantitatively and qualitatively.

4.7.1 Case study

To validate our approach, we have recreated the animation of a well-known scene from Robert Zemeckis' movie "Back to the future". This short (80 seconds) scene is a moderately complex interior scene, with four main characters, all engaging in a variety of actions, including two-way and three-way dialogues, physical contacts, and everyday activities such as sweeping the floor and serving food. To recreate this scene, we had to go through several stages of the movie-making process. We first retrieved the script of the movie (Figure 4.19) and translated it into a sequence of actions annotated to provide $(subject, verb, object)$ descriptions at the right time-codes (see section 4.2).

MARTY (in disbelief): You're George McFly.
GEORGE: Yeah, who are you?
GOLDIE: Say, why do you let those boys push you around like that?
GEORGE: Well, they're bigger than me.
GOLDIE: Stand tall, boy, have some respect for yourself. Don't you know that if you let people walk all over you know, they'll be walking all over you for the rest of your life? Listen to me, do you think I'm gonna spend the rest of my life in this slop house?
LOU (has heard the remark): Watch it, Goldie.
GOLDIE (he's on a roll): No sir, I'm gonna make something out of myself, I'm going to night school and one day – I'm gonna be somebody.
MARTY: That's right, he's gonna be mayor.
GOLDIE: Yeah, I'm– mayor. Now that's a good idea. I could run for mayor.
LOU: A colored mayor, that'll be the day.
GOLDIE: You wait and see, Mr. Caruthers, I will be mayor and I'll be the most powerful mayor in the history of Hill Valley, and I'm gonna clean up this town.
LOU: Good, you could start by sweeping the floor.
GOLDIE (to himself): Mayor Goldie Wilson, I like the sound of that.
MARTY: Hey Dad, George, hey, you on the bike.

Figure 4.19: *Extract of the script from* Back To The Future

After modeling each of the characters (Figure 4.20) and the environment (Figure 4.21), we animated them to match the annotated actions. In order to further improve the quality of the animation, we also performed lip syncing with the dialogues using *Faceshift*. Finally, we set up the lighting and manually placed twenty-five cameras for the whole duration of the sequence (sixteen of them closely approximating the actual cameras from the original movie, and nine providing alternative angles). In chapters 5 and 6, we use different camera planning techniques to provide this input to the system.

Once all these stages were completed, to conduct the experiments of the following sections, we generated many variations of the same sequence using our system and, for each, performed a last post process operation to link the video with the audio track of the movie.

Finally, for evaluation purposes, we are making our experimental data (including rushes and their annotations) and our experimental results publicly available[1]. This dataset also contains a working Unity3D project with all the 3D models and animations.

[1] https://team.inria.fr/imagine/continuity-editing/

(a) *Marty* **(b)** *George* **(c)** *Goldie* **(d)** *Lou*

Figure 4.20: *Characters of* Back to the future.

(a) **(b)**

Figure 4.21: *3D model of the scene from* Back to the future *viewed from inside (a) and outside (b) the bar.*

4.7.2 User-study

To demonstrate the soundness of our model, we have experimentally compared our method (O) to the original edit of the scene (Z) reproduced from Zemeckis' movie (which serves as a reference for comparison with expert cinematographers) and to three degraded versions: a degraded version (Ds) where the content of shots is not considered (*i.e.* the shot cost is removed), a degraded version (Dp) where the enforcement of the specified cutting rhythm is not considered (*i.e.* the rhythm cost is removed), a degraded version (Dc) where visual discontinuities are enforced (*i.e.* the cut cost is reversed).

We performed a subjective evaluation of our method by designing a perceptual user-study. Twenty-one participants volunteered for this experiment. They were 27.5 ($\pm$ 5.7) years old (range: 20 to 42). They were naive with respect to the purpose of the experiment. All had normal or corrected-to-normal vision. They gave written and informed consent and the study conformed to the declaration of Helsinki. We prepared 5 stimuli (a fully edited version of 80 seconds per method). Participants were asked to observe the video stimuli while seated in front of a desk. After each stimulus viewing, participants were asked to rank the global film-making quality[2] on a discrete scale ranging from 0 (very bad) to 10 (very good). In total, they repeated this task 20 times (5$\times$4 repetitions). Stimuli were presented in a randomized order. The total duration of the experiment was about 30 minutes.

[2]We additionally proposed a number of criteria that participants could consider to score each version: the enhancement of characters performance, the synchronization of cuts with the scene content, the aesthetic of shots.

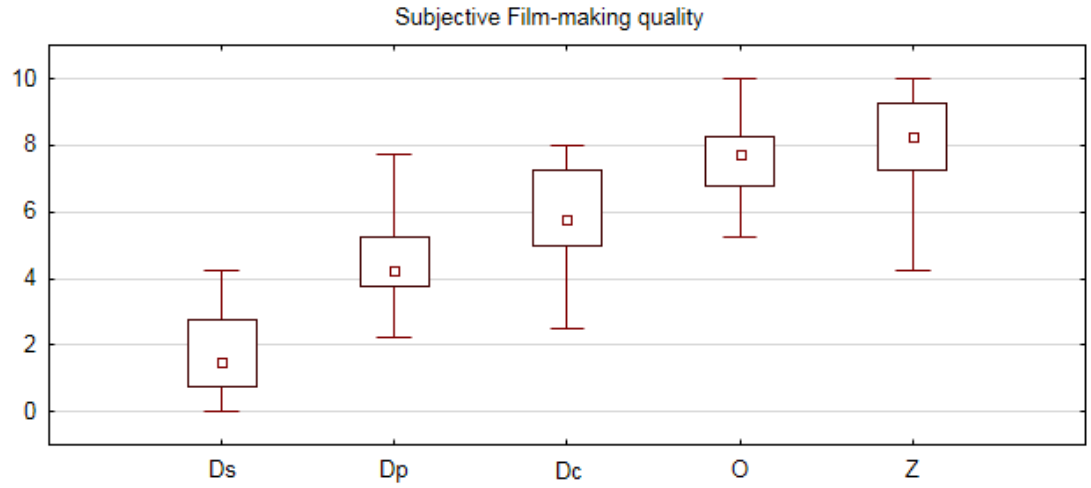


Figure 4.22: *The scores obtained by the five stimuli, each represented with a whisker plot. The central point represents the median score, the box represent the scores between the first and third quartiles, and the bottom and top lines represent the minimum and maximum scores.*

We tested three hypotheses. **H1:** editing has an impact on the perceived quality of the observed video stimulus; **H2:** each of the three terms in our cost function (shot content, continuity rules and cutting rhythm) has a positive impact on perceived quality; **H3:** the perceived quality of the version done by an expert cinematographer is significantly higher than our method. The dependent variable in this study was the participants' score given to each version. Figure 4.22 illustrates the scores obtained for each version.

Hypothesis H1 was confirmed using a non-parametric Friedman test ($p < 0.05$). Post-hoc comparisons summarized in Figure 4.22 confirm hypothesis **H2** that the consideration of each of the three key aspects has a positive impact on subjective quality, but discard hypothesis **H3** that the version done by an expert cinematographer is significantly better scored than the one generated by our system (though Zemeckis' version globally obtained better scores).

This "subjective analysis" proves the validity of the method but does not reveal limitations nor provide any lead for future work.

4.7.3 Qualitative comparison

To further validate our approach and highlight areas of improvement we now proceed to an extensive qualitative comparison over the detailed criteria. As mentioned before, no ground truth exist in film-editing – different editors would create different, yet correct, versions of the same sequence – but a precise comparison of our results with the original movie offers precious insights of the decision process. In the following, we analyse the differences between an automatically generated edit and the original sequence of the movie.

Figure 4.23 illustrates the two edits and will be used as reference for the rest of this analysis. We observe that 35% of the shots are shared by the two edits. Thus, 65% of the time, the director and/or editor took a different decision. To better understand these differences we now present the detailed comparison of the three aspects of editing: shot selection, cut and rhythm of the sequence. The first aspect is the core of the shot selection process, and so we detail it more extensively by looking at both the action visibility and the hitchcock principle. We then

analyse the main cutting decisions and “mistakes” made by R. Zemeckis that are detected by our system. Finally, we compare the shot durations of the two sequences.

Figure 4.23

Figure 4.23: *Comparison of Human and computer generated edits of the same sequence from 25 cameras. Each camera is assigned a color based on its id. The scenario describes the actions involving Marty* (M.)*, George* (G.)*, Goldie* (Go.)*, Lou* (L.) *and the Cashier* (C.).)

For this analysis, we computed the cost of each criterion for the two edits. To highlight the differences we display these costs using the colormap in Figure 4.24 (blue for a low cost and red for a high cost).

Figure 4.24: *Colormap used to display the cost values. Lowest cost* (0) *are displayed in blue and highest cost* (1) *in red.*

4.7.4 Qualitative evaluation of the Action Visibilty

The first analyzed criterion is the action visibility. Figure 4.25 highlights several significant differences in action visibility between the original and generated sequences.

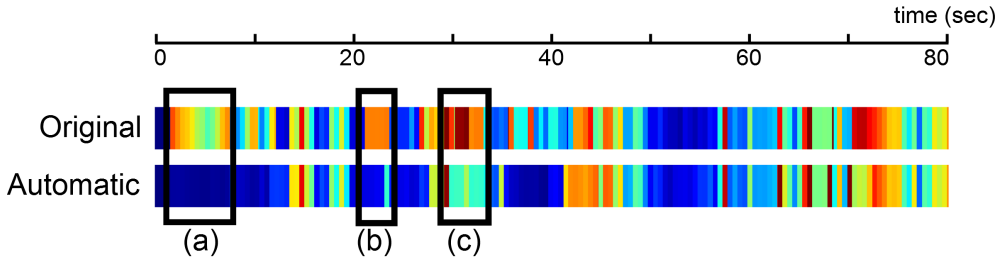


Figure 4.25: *Action visibility costs computed throughout the whole sequence for the original movie and the automatically generated sequence. Main differences are highlighted in (a),(b) and (c) where the visibility of the characters in Zemeckis' version is bad.*

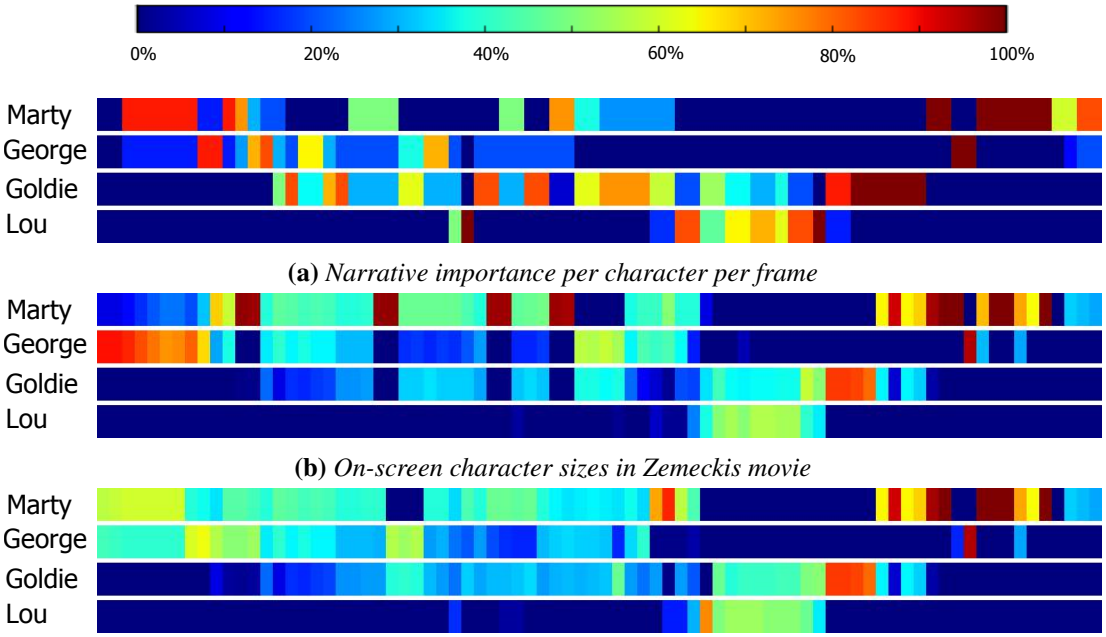
Figure 4.26 illustrates the difference in visibility highlighted in Figure 4.25(a). At this frame, the only occurring action is Marty, staring at George. The lack of visibility on Marty's face was obviously orchestrated by the director in order to slowly reveal Marty's reaction. While our system safely chose a shot with the proper visibility over the two characters for the whole duration of the action (see Figure 4.26b), R. Zemeckis uses this lack of visibility to drag the audience's interest toward Marty's appearing face and emphasize his reaction.

(a) *Shot from the movie*

(b) *Computed shot*

Figure 4.26: *Shot taken at* $t = 3s$ *when Marty stares at George. Zemeckis uses the lack of visibility to drag the audience's attention (a). The generated sequence uses a shot with perfect visibility over the two characters (b).*

When trying to automate a process as complex as video editing, one is bound to make simplifying assumptions. Even though assuming that poor visibility is synonymous with poor



(a) *Narrative importance per character per frame*

(b) *On-screen character sizes in Zemeckis movie*

(c) *On-screen character sizes in our solution*

Figure 4.27: *The relative narrative importance of a character is computed from all actions where it plays a role. It correlates with the screen size of the characters in both the Zemekis movie (linear correlation coefficient of 0.56) and our solution (0.73). Here the colormap is not related to the computed costs.*

shot quality might sound reasonable, in some circumstances, it might not be the case. With this optimization based approach, the goal is only to avoid making mistakes. Handling such a motivated and complex shot would require a lot more reasoning on the actions and computation of importance.

4.7.5 Qualitative evaluation of the Hitchcock Principle

The other important criterion used in shot selection is the action ordering. It is based on the Hitchcock principle detailed in section 4.3.3. Figure 4.27 shows the evolution of the computed importance of the characters and their respective size on the screen during the sequence. It reveals noticeable differences between the two. Figure 4.28 highlights these strong differences with the hitchcock principle's cost.

The shots in Figures 4.26a and 4.26b also illustrate the difference (a) of Figure 4.28. At this specific moment in the story, Marty is the most important character, followed by George. In Figure 4.26a, it is obvious that the narrative importance of the characters does not match their screen sizes, as Marty barely appears in the screen. During this shot, the cost slowly decreases with the appearance of Marty in the frame which slowly reaches a "Hitchcock equilibrium". This example illustrates one of the current limitations of the system. It does not allow this form of intensification. It would require a variable importance within the action itself. Figure 4.29a offers another illustration of Hitchcock principle violation (see Figure 4.28(b)). It is taken during a dialogue between George and Goldie and yet most of the screen space is occupied by Marty in the foreground. Figure 4.29b shows the automatically selected shot. This one satisfies the hitchock principle as it focuses on the two characters involved.

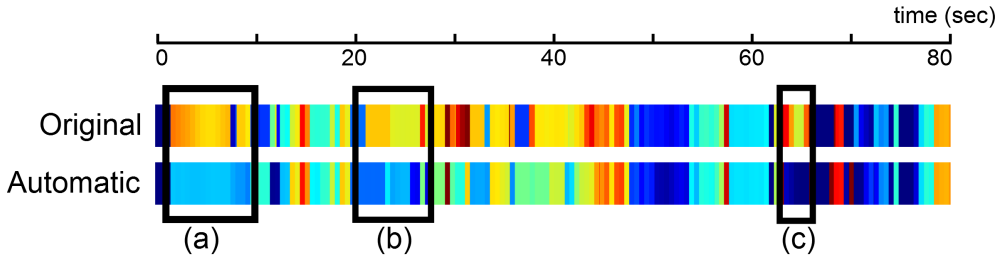


Figure 4.28: *Hitchcock costs computed through the whole sequence for the original movie and the automatically generated sequence. Main differences are highlighted in (a),(b) and (c) where the generated version has a better* Hitchcock equilibrium.

(a) *Shot from the movie* **(b)** *Computed shot*

Figure 4.29: *Shot taken at $t = 23s$ when Goldie talks to George.* Zemeckis *included the main protagonist in the frame (a). Only characters involved in occurring actions appear in the generated version (b).*

Here Marty is not involved in this specific dialogue, but his presence on the screen is important since he is the main protagonist of the movie. It shows that he is listening to the conversation and, thus, gives information to the audience on his understanding of the situation. This limitation does not come from the Hitchcok principle but rather from the computation of importance itself. It does not consider any higher level of importance or involvement in the situation (such as a three person dialogue) or the global story.

Finally, in order to validate our implementation without depending on the interpretation of a scenario, we make the assumption that Zemeckis did follow the Hitchcock principle. We then compute the importance of the characters from their actual onscreen sizes in the movie (see Figure 4.3.1(b)). Using this importance as input to our system instead of the scenario, the computed edit then shares 75% of its shots with the original version (instead of 35% when using the scenario). To further validate our implementation, we computed an edit by only relying on Hitchcock principle, with no regards to any other rule. This experiment proves the validity of our implementation as the resulting edit shares over 95% of its shots with the original. The remaining 5% are due to the time granularity used for this test; shots are only evaluated every half second to reduce computational time. Figure 4.30 details the different edits computed.

4.7.6 Qualitative evaluation of the cuts and continuity editing

In this section, we analyse the quality of the cuts with regards to the continuity editing style. Figure 4.31 shows the computed costs of each cut (the value displayed for each shot is the cost of the previous cut). This cost is a weighted sum of costs computed from the different

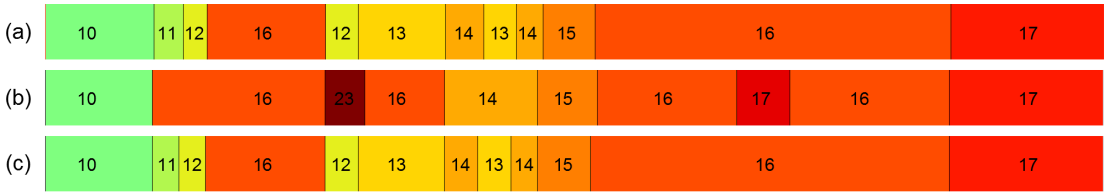


Figure 4.30: *Comparison of the original movie (a) with two other edits that uses narrative importance extracted from the movie: one uses the whole algorithm (b) and the other one only uses the hitchcock principle.*

continuity rules mentioned in section 4.4 with an emphasis on the left-to-right continuity and jump-cut rules.

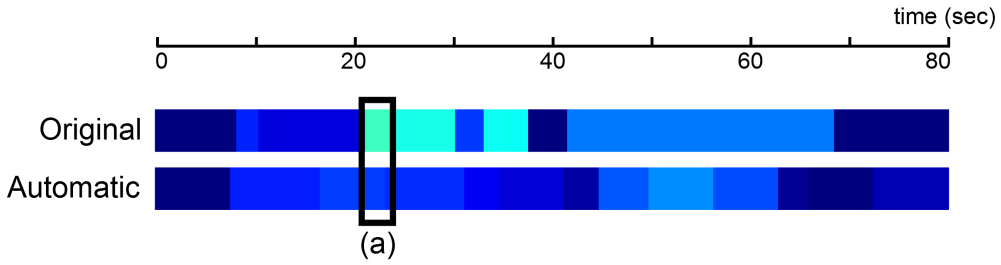


Figure 4.31: *Cut costs computed through the whole sequence for the original movie and the automatically generated sequence. In both version, only minor transgressions can be noticed (a)*

For both the original and automatic editing, only minor transgressions can be noticed, as illustrated in Figure 4.32 with the spatial displacement of Marty in screen space. None of the two sequences have jump-cuts or left-to-right discontinuity.

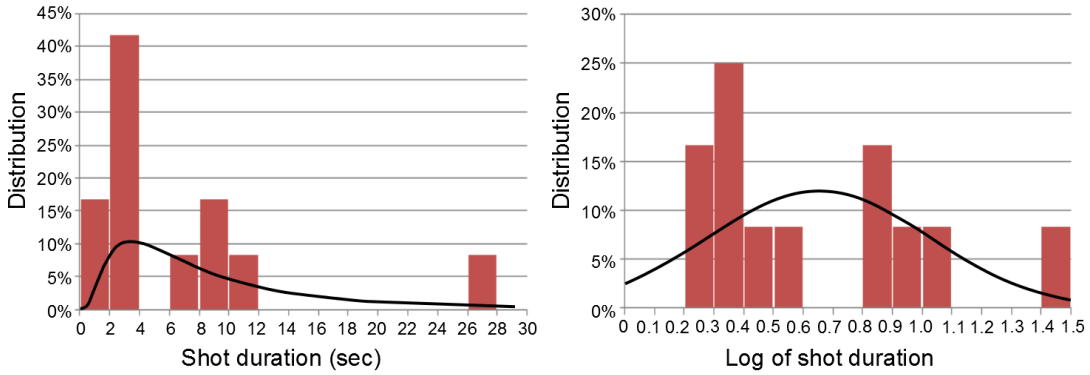
Figure 4.32: *Cutting discontinuity: the position of* Marty *significantly changes from one shot to another, introducing a position discontinuity.*

This result not only confirms that the original sequence from *Back To The Future* satisfies the rules of continuity but also that the optimization based approach gives a proper implementation of the continuity editing style for this dataset.

4.7.7 Qualitative evaluation of the pacing

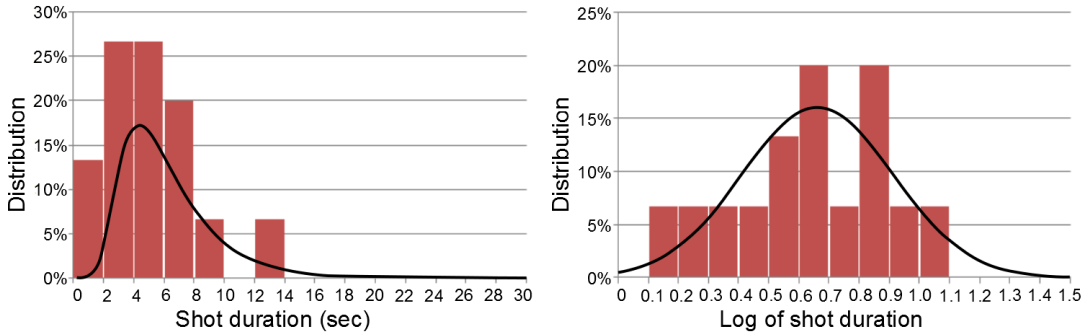
Finally, the last element of film-editing that we analyse and compare with the original sequence of *Back To The Future* is the rhythm. For the generated sequence, an average shot length of 5.25s was used as parameter of the cost function which resulted in an actual ASL of 5.31s. In

the original version, the average shot length (ASL) is 6.6 seconds and the distribution of shot lengths is well approximated with a log-normal law of mode $m = 2.28$ and standard deviation $\sigma = 0.82$. Figures 4.33 shows the distributions of the shot durations for the two versions.



(a) *ASL = 6.64s; SD = 0.82*

(b) *Mean = 0.66s; SD = 0.37*



(c) *ASL = 5.31s; SD = 0.51*

(d) *Mean = 0.66s; SD = 0.25*

Figure 4.33: *Shot duration distributions for the original version (a) and the computed edit (c), along with the distribution of the log of their shots durations respectively (b) and (d).*

Despite the two distributions being similar and both relatively close to the computed log-normal distribution (over the whole sequence), the cumulative cost of the pacing is four times larger with Zemeckis' version than our automatically generated sequence. The explanation appears in Figure 4.34, which shows the computed pacing cost for each shot of the sequence. Two categories of "bad" pacing can be identified in the graph: very short takes and very long takes. The high cost highligted in Figure 4.34(a) is due to a very small shot duration (see Figure 4.35(b)). This shot breaks the rhythm of the sequence to show the short reaction of the character. In the automatically generated version, the same sequence is handled using two longer shots that cover several actions.

In Zemeckis' movie, the last two shots of the sequence are long takes lasting 27 seconds and 11 seconds with elaborate panning camera motion (see Figure 4.37). Due to their deviation from the ASL, the cost of these two shots is very high, as shown in Figure 4.34(b). In the computer-generated version, those same 38 seconds are handled with ten different shots from six different viewpoints. This gives a different dynamic to the scene, but does not make it a better solution. By preventing large deviation from the ASL, the computer-generated version sometimes fails to find better solutions. Future work is needed to better understand how to han-

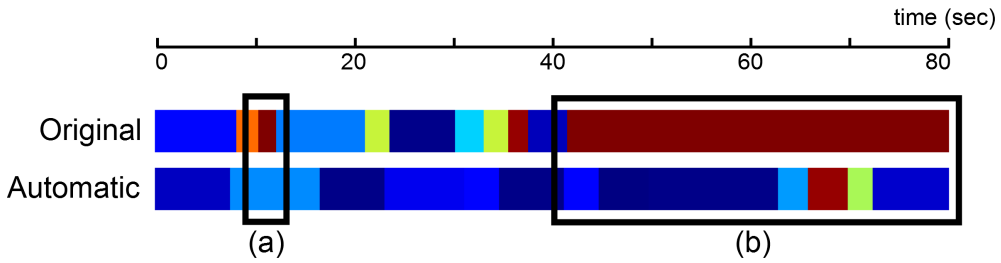


Figure 4.34: *Pacing costs computed through the whole sequence for each shot for the original movie and the automatically generated sequence. Important costs are computed in the original version for very short (a) and very long (b) shots.*

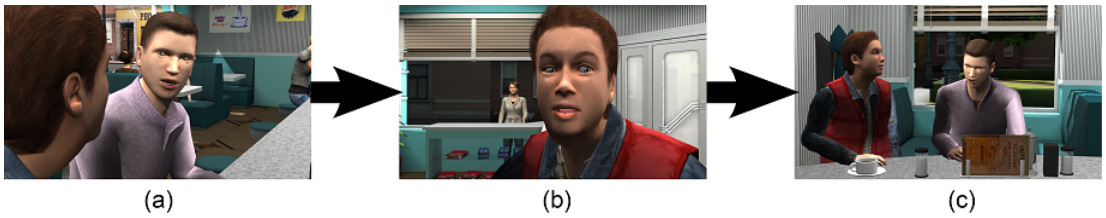


Figure 4.35: *Shot sequence from the original movie. A very short shot (b) is inserted to show* Marty*'s reaction.*

Figure 4.36: *Shot sequence from the generated version. Only shots with all characters involved in the occurring actions are used.*

dle such cases, where the quality of extended shots with elaborate camera movements should probably be given more weight.

Figure 4.37: *Very long take from the original movie handled with an elaborate panning camera motion.*

To further illustrate and validate our method, we have also generated two new versions of the scene with ASLs of resp. 2 seconds (fast cutting style) and 10 seconds (slow cutting style). The choice of shots is very different in those different styles. In fast cutting style, there is a preference for close-ups and medium shots. In slow cutting style, there is a preference for

medium long shots and full shots. Figure 4.38 shows the first 40 seconds of the two sequences. The complete videos are also available in the dataset.

Figure 4.38: *Comparison of the first 40 seconds of two sequences generated with ASL of two seconds for the left edit and ten seconds for the right edit. Each camera is assigned a color based on its id (in this figure, colors are not related to the cost of the shots).*

4.7.8 Framework and implementation details

To conduct all the previous experiments and test our results, we implemented a complete framework that uses our method to perform the editing of animated content. This framework was fully implemented in C# with the 3D engine *Unity3D*. It takes as input a 3D animated scene along with a scenario file and requires some quick setup to match the elements of the scenario with the 3D elements of the scene. Once the setup completed, the framework guides users through the different steps (see Figure 4.39) to transform the raw animation into a fully edited sequence:

1. **Generate the cameras:** this first step was not addressed in this chapter. At this stage, we simply use predefined cameras. The work detailed in chapters 5 and 6 offers interesting solutions to automatically generate camera shots.

2. **Generate the rushes:** the system performs the computation of the costs for each camera and for the whole duration of the animation. It also computes the costs for every possible transition in the film graph. Users have access to this information through an interface similar to Figure 4.41.

3. **Compute the editing:** this is the core of the system. It performs the editing using the costs computed in the previous step. Default parameters can be modified to generate different outputs.

4. **Watch the results:** it plays the generated sequence. Users still have access to the information previously computed and can loop back to the previous step if not satisfied with the results.

Figure 4.39: *Main menu of the system. It guides users through the*

Figure 4.40 shows the interface implemented to allow users to adjust the default parameters of the method. By modifying the weights of the cost functions and providing different values for the desired pacing, users define an editing profile specific to their own editing style. The feature highlighted in the figure offers users a finer control of the final output. It allows them to force the selection of a shot at any moment in time.

Finally, Figure 4.41 shows the main interface of the system. It is used after generating the rushes to let the user analyse the shots and after computing the edit to replay the solution. The interface is divided into three part. The first one is made of a navigation bar and the set of all

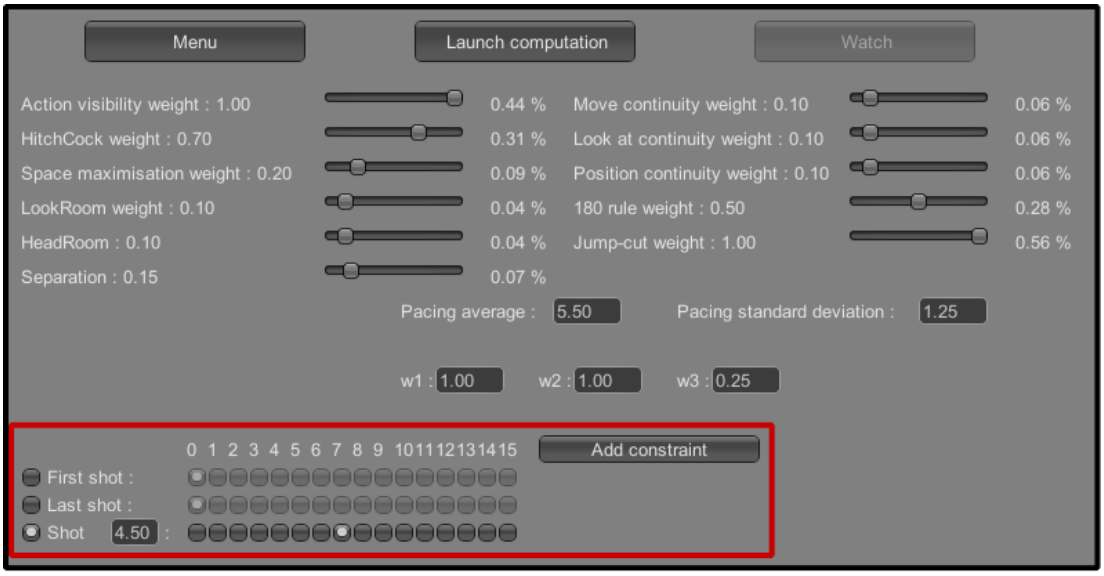


Figure 4.40: *Graphical interface for the computation of an edit. Users can adjust the weights of every cost functions. They can also specify the desired pacing and add constraints on shots.*

available cameras. Several modes can be activated to display bounding boxes or switch to a single camera view. The second part contains the scenario. It shows every actions unfolding in the story – past, current or future. The last part of this interface displays all the information on the selected shot and the previous cut. It details the costs associated with each cinematographic rule to help users adjust their editing profile if needed.

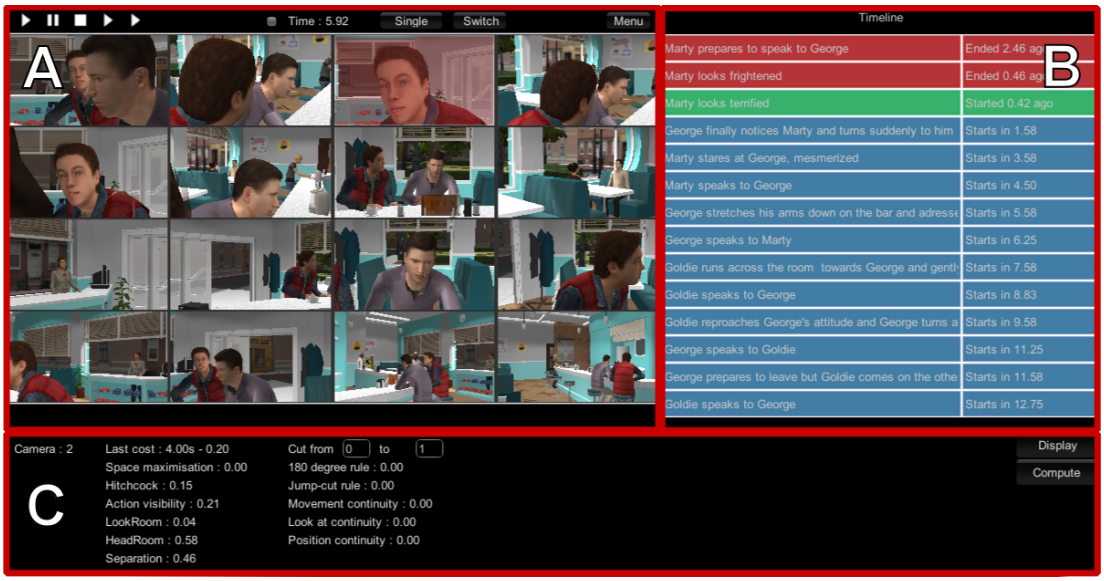


Figure 4.41: *Main interface of the framework. It is made of three parts. (A) displays all generated shots and highlight the selected one. (B) shows the scenario with the past (red), current (green) and future (blue) actions. (C) displays all the information on the selected shot and the previous cut.*

Even though the interface could be improved to be more user friendly, this framework represents the first draft for a smart digital assistant to the production of edited videos.

4.8 LIMITATIONS AND FUTURE WORK

Our model remains currently limited in several aspects. First, it is currently limited to the case of linear editing, where the chronology of events is maintained. This limitation could be addressed by allowing temporal ellipses and re-ordering of events. In addition, the comparative study conducted in section 4.7 showed that the sequences generated by the system minimize violations of editing and cinematographic rules at the expense of stylistic decisions. The proposed model only enables to control the pacing. Other style parameters such as shot composition (flat vs. deep staging), camera movements (static shots, dolly shots, crane shots), transitions (reverse shots) and lighting preferences would favor user creativity. Our model of shot selection is based on bounding boxes of the character's body parts and a primitive classification of their actions. Objects and places in the scene, as well as character's moods and intentions, should also play a part. Furthermore, we should note that the semi-Markov assumption has limitations of its own. Important film editing patterns such as book-ending, separation or parallel action [Sha82] cannot be taken into account by such a memory-less model. The investigation of higher-order Markov models or context-free grammars could be pursued to overcome such limitations.

In contrast to previous work in virtual cinematography, our model clearly separates the roles of the virtual cinematographer and the virtual film editor. This has important consequences for future work. Firstly, it makes it possible to combine our film editing model with any given cinematography (camera control) technique in a generate-and-test approach where the virtual cinematographer proposes camera choices, and the virtual film editor chooses the best available combination. Secondly, the evaluation of our cost function can be performed in screen space, and does not require knowledge of the three-dimensional configuration of actors in the scene. As a result, it is possible, at least in principle, to learn the parameters (weights) of our model from examples of real movies. This opens the way to extrapolate our model to other editing styles beyond the basic continuity editing style studied in this paper. Future work is needed for annotating movie scene examples with actions and their narrative importances, and learning statistical models of film styles from those training sets. A promising approach in this direction is to automatically classify shots in real movie scenes using plan recognition methods [CPR98, CPR00] based on computer vision techniques for recognizing actors' shapes, appearances, motions and behaviors [HFR06]. Alternatively, one may use movies scripts [RT03, Ron04, CJMT08, PGHA15] or descriptive video services [RRTS15] as an external source of high-level annotation, allowing to train different film editing models from real movie scenes in different styles.

Finally, at this stage, the main limitation of the system remains its lack of consideration for the cinematography aspect. It does not deal with the automatic computation of shots and is restricted to a limited choice of cameras. Combined with virtual cinematography, our approach promises to significantly extend the expressiveness and naturalness of automatic virtual movie-making. In chapters 5 and 6, we address this current limitation using our previous work on camera control to optimize over camera positions and framings and tackle the problem of virtual movie-making as a whole.

4.9 SUMMARY

In this chapter, we have presented a continuity-editing approach to the automated creation of cinematographic sequences for 3D animated contents. We have introduced the notion of editing graph and showed how dynamic programming can be used to compute an optimal edit under a semi-Markov hypothesis. We also provided a thorough description of means to rank shots and edits, and to measure the distance to a specified cutting rhythm. Our solution is supported by subjective evaluations obtained in a perceptual user study as well as a thorough qualitative comparison with a professionally edited sequence. The proposed approach performs a clear shift from existing techniques such as idiom-based representations, with a level of expressiveness not addressed by previous contributions. Finally, this work provides the foundations to address novel challenges in automated cinematography, such as learning and reproducing cinematic styles from real-movies.

CHAPTER

5

NARRATIVE-DRIVEN CAMERA CONTROL FOR CINEMATIC REPLAY

After addressing separately the cinematography and editing aspects of virtual movie-making, we now tackle the overall challenge. In this chapter, we present a system that automatically generates cinematic replays. It exploits the narrative and geometric information present in games and computes camera framings and edits to build a cinematic replay of the gaming session.

Our solution relies on Alfred Hitchcock's principle, using the narrative importance of the characters in the story to guide the framing and editing process. We propose a virtual director that maps narrative importance of characters into camera behavior specifications, and we then devise a technique to transform the camera behavior specifications into shots using different camera steering techniques introduced in chapter 3. We demonstrate our system by implementing a Director that controls three camera behaviors (one for master shots, one for shots on the player character, and one for reverse shots) and test it by interfacing with a full-fledged serious game (*Nothing for Dinner*).

5.1 INTRODUCTION

In this chapter, we apply the work detailed in chapters 3 and 4 to address the challenge of cinematic replays: computing a cinematographically aesthetic review of the animation generated during a game session. With the continuous growth in popularity of video games, gaming companies have invested a lot to improve the cinematographic quality of their products. Cinematics and cinematographic techniques now play a significant role to enhance the gaming experience [PKG12]. Furthermore, with the advent of multi-player games, and the possibilities of sharing players' performance and playing experiences on the web, there is a significant demand in generating relevant cinematic replays of gaming sessions. Dedicated tools have been designed to ease the creation of replays[1], either to report game experiences, or for more aesthetic considerations such as machinima.

However, a close look at these dedicated tools shows that a lot is still done manually, typically in selecting the appropriate moments, setting the cameras, and performing edits between multiple cameras. In parallel, as detailed in section 2.2, researchers in computer graphics focusing on automated virtual camera control have been proposing a number of efficient techniques to automatically place and move cameras as well as editing algorithms to automatically or interactively edit the shots of a movie. Moreover, unlike the work conducted in chapter 3, the creation of cinematic replays is not subject to realtime constraints, thus allowing the use of offline planning techniques.

These approaches are mostly founded on what could be referred to as "action-based" camera control, in the sense that a typical idiom is associated to each action occurring in the 3D environment (an idiom is a stereotypical way of shooting the action, either through a single shot or a sequence of shots). A film is then constructed by computing the best sequence of shots portraying a sequence of actions performed by the characters (as in [ER07, LCRB11a, LCCR11, MKSB11]).

Automated cinematography techniques have rarely been adapted to the specific case of cinematic replays (with the notable exception of [DYR11a]). The problem is actually challenging. Character tracking techniques such as [HHS01, LC12] would generate cinematics of low interest by creating continuous camera motions without cuts. Idiom-based techniques [CAH*96] would typically fail due to the inability to handle complex situations and the necessity to design idioms for many different actions and situations. Finally, optimization-based approaches such as [ER07] require the manual specification of cinematographic patterns for each situation, while [LCCR11] maps actions to shot preferences in a straightforward way.

To overcome limitations of idiom-based techniques, as well as approaches which solely rely on characters' actions, we propose a more principled approach. Based on Hitchcock's well-known rule which states that *the size of a character on the screen should be proportional to its narrative importance in the story* [TS67, Haw05, DeL09], we propose means to compute the individual importance of each character from the replay, and map these importances with cinematographic specifications. Importance therefore serves as an novel intermediate representation which can account for more elaborate and contextual situations between characters in a replay sequence, typically including intentions, significance of the characters in the whole story, as well as causal relations between events.

Unlike idiom-based techniques, this approach to cinematography is agnostic to the type of action occurring. It only requires the provision of an importance function on the characters. The mapping between the importances and the camera specifications is not related to how im-

[1] see Simatography for the Sims, Warcraft movies, replay editor or Team Fortress 2 or Total war shotgun 2

portance is computed, therefore providing an independent and reusable set of cinematography techniques.

Our approach comprises a preliminary stage which enables the extraction and computation of the characters' importances from a game trace (this is specific to each game engine). Our technique is then composed of three stages: (i) mapping importances with cinematographic specifications by defining camera behaviors, (ii) animating cameras by enforcing the specified behaviors, and (iii) editing the rushes computed by the cameras.

The contributions of this chapter are: (i) a character importance-based approach to drive camera placements and camera edits, thereby moving a step beyond action-based and idiom-based techniques, (ii) a novel incremental technique to convert camera specifications into camera coordinates using spherical and toric surfaces, and (iii) a smooth camera animation technique that maintains the specifications as the scene is evolving and enables smooth transitions between different camera specifications.

The benefit of the system stands in its ability to effectively convey, with a realistic and dynamic cinematic style, a dialogue-based video game session through a collection of simple and dynamic camera behaviors.

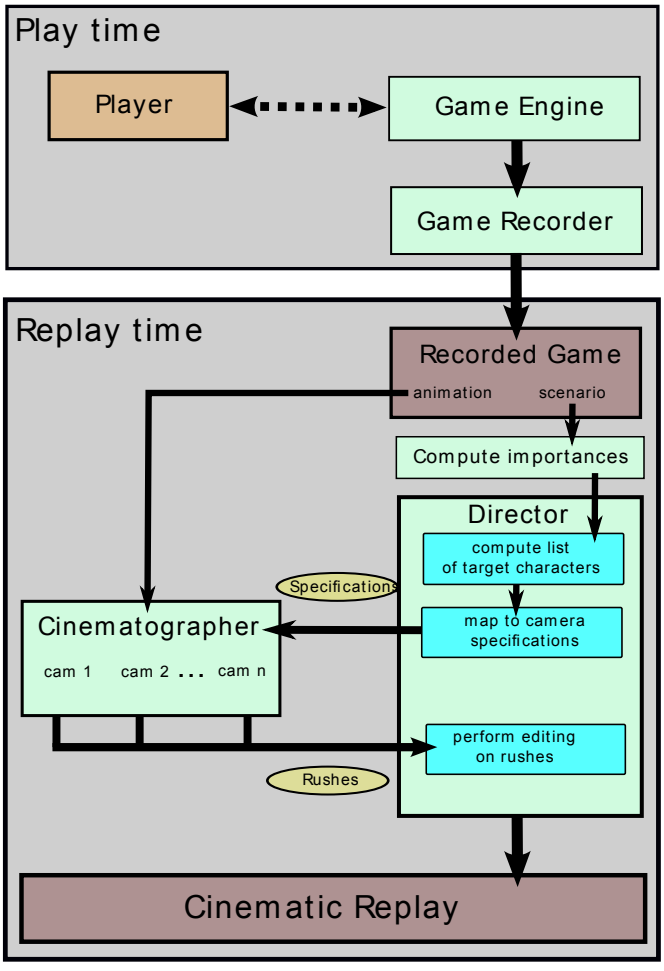


Figure 5.1: *System overview*

5.2 OVERVIEW

In this section, we give an overview of the method used to generate the cinematic replay from the extraction of the information to the generation of the camera rushes. In order to produce a cinematic replay, we need to access to all the information generated by the game engine. In our case, we devised a game recorder with two purposes: record the animation of the scene (characters, objects, etc.) and retrieve the scenario from the game engine.

Figure 5.1 shows the different stages of our system and Figure 5.2 illustrates them with a concrete example. Our system makes use of the recorder with two other components: a virtual director and a virtual cinematographer. Our director's goal is to extract the important targets using the narrative importance of the characters (see Section 5.3, and then assign camera specifications to the cinematographer using the prose storyboard language (PSL) introduced by [RGB13] (see Section 5.4).

It is then the task of the cinematographer to place and move different cameras in the scene. The cinematographer transforms the high-level PSL specifications given by the director into 3D coordinates, and angles for the camera using the geometric information on the scene (see Section 5.5).

Finally, once all the cameras have been properly animated for the whole replay, the cinematographer sends back to the director the rushes filmed by the cameras. The director is then in charge of performing the editing and creating the final cinematic replay as presented in our results.

5.3 AN IMPORTANCE-DRIVEN APPROACH

In our context, we assume that the game engine is based on a classical two-level narrative structure: beats that describe a narrative unit [MS02], and atomic actions that compose a beat (*e.g.* characters moving, speaking or reacting). This is not a restrictive hypothesis nor a prerequisite, and typically can be adapted to whatever information is provided by the game engines.

Using this two-level structure, we compute two different levels of importance. The first level of importance $I_{beat}(c,t)$ provides an estimation of the importance of character c over a specified beat, measured in terms of how many actions he was involved in over the beat occurring at time t and the significance of his role in each action (significance is a numerical value mapping the range "absent" to "significant" to 0..1). The second level, $I_{atomic}(c,t)$ provides an estimation of the importance of a character c from the relevance of the action he's involved in, and from the significance of his role in the action.

$$I_{beat}(c,t) = \sum_{a \in \mathcal{A}_{beat,c}} S_c(a,t)$$

$$I_{atomic}(c,t) = \sum_{a \in \mathcal{A}_{c,t}} R(a,t) \times S_c(a,t)$$

where

- $\mathcal{A}_{beat,c}$ is the set of actions performed by character c in the specified beat;
- $\mathcal{A}_{c,t}$ is the set of actions performed by character c at time t (a character can perform multiple actions simultaneously);

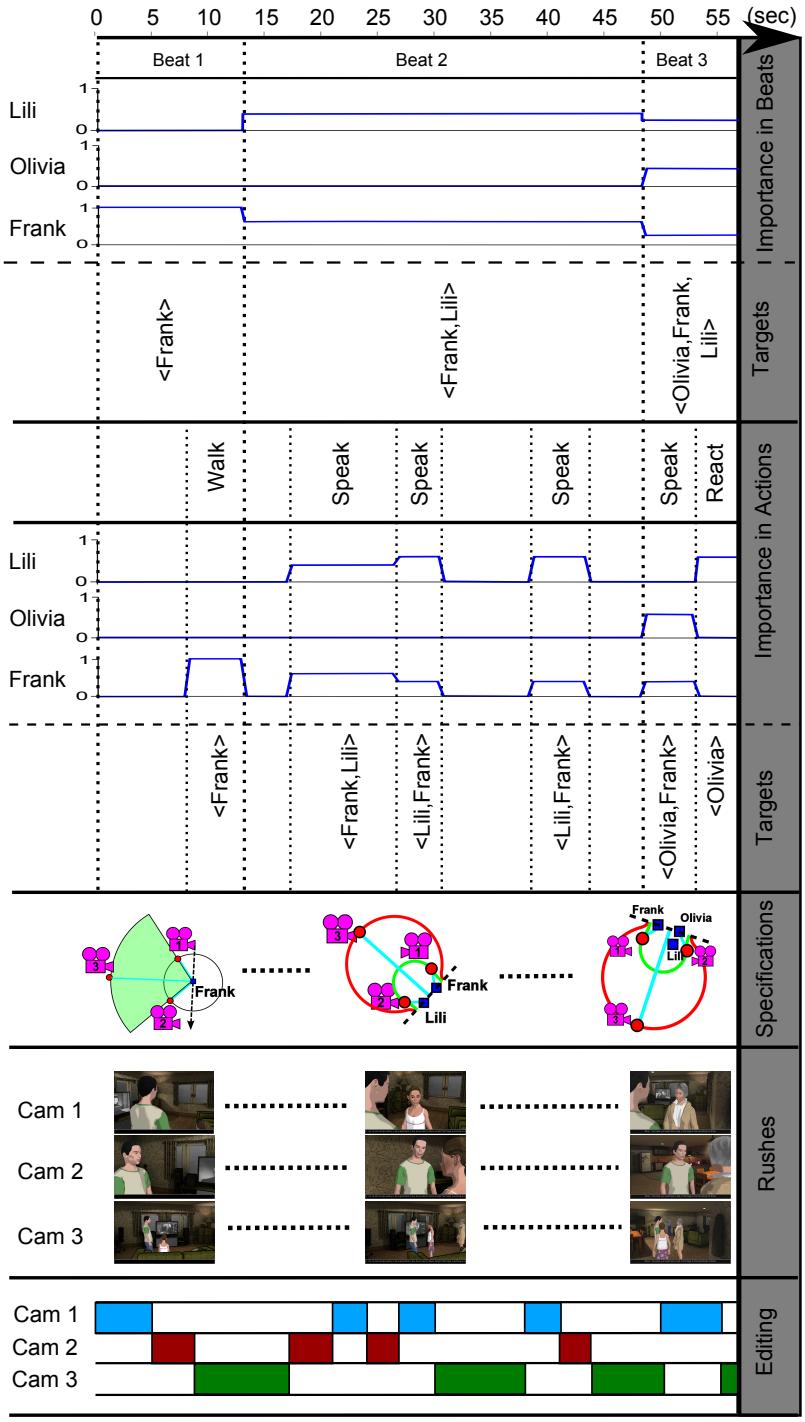


Figure 5.2: *An illustration of the complete cinematic replay process. Starting with the list of beat and actions, the importances of the characters are computed to establish the configurations of characters. Each configuration is converted into camera specifications given each of the camera behaviors (3 behaviors are defined here). Finally, the rushes are generated from the specifications and edited.*

- $R(a,t)$ is the relevance of the action a performed at time t with regards to the completion of the overall objective of the game;
- $S_c(a,t)$ is the significance the role played by the character c in action a at time t.

Values of $R(a,t)$ and $S_c(a,t)$ are considered to be provided by the game engine. While other game engine may provide different information, the key here is to propose a mapping from this information to importances.

5.4 The Director: from importance to specification of camera behaviors

Once the importances have been computed for each character, we propose to map this list of importances with a collection of camera behaviors. The purpose of this collection is to simultaneously control multiple individual cameras to offer simultaneous distinct viewpoints over which film editing is performed.

5.4.1 High level specifications

In order to specify camera behaviors, a shot description language is necessary. It defines the specification that will be associated with the characters' importances.

The *Prose Storyboard Language* (PSL) elaborated by [RGB13] seemed appropriate for this task as it defines the syntax and semantics of a high-level shot description language. Using PSL, partial or full camera specifications can be authored (*i.e* expecting only one solution or a set of possible solutions). Figure 5.3 shows the subset of the language we focus on. We extended the grammar to handle Point Of View (*POV*) shots (a shot from the physical point of view of the character).

```
<Composition> ::= [<angle >|<pov>] {<FlatComposition>}+
<FlatComposition > ::= <size > on <Subject >[<profile >]
                       [<screen >] (and <Subject >[<profile >]
                       [<screen >] [in ( back | fore )ground]) *
<Subject> ::= (<Actor>|<Object>)+
<angle> ::= ( high | low ) angle
<size > ::= ECU | BCU | CU | MCU | MS | MLS | FS | LS | ELS
<pov> ::= POV (<Actor>|<Object>)
<profile > ::= 3/4 left back | left | 3/4 left | front
               | 3/4 right | right | 3/4 left back | back
<screen> ::= screen ( center | left | right )
```

Figure 5.3: *PSL Grammar - Screen composition*

5.4.2 Behaviors

In order to ease the mapping between the characters' importances and the camera behaviors, we propose to abstract the characters as either PC (player character), P_i (primary non-player

characters), and S_i (secondary non-player character). PC is directly provided by the game engine, while P_i and S_i are derived from the relative importance of the characters. We manually determined two threshold values α_S and α_P. At each frame, all the characters (but the player character) with an importance higher than α_P are considered primary characters. The remaining ones with an importance higher than α_S are considered secondary characters. All the others are neglected.

This abstraction creates a configuration of characters for each frame. The different configurations of characters are displayed in Table 5.1.

Configuration	Meaning
$< PC >$	The player character is the only target
$< P_0 >$	One primary target that is not the player character
$< PC, P_0 >$	Two primary targets, one of which is the player character
$< P_0, P_1 >$	Two primary targets not including the player character
$< P_0, S_0 >$	One primary and one secondary target not including the player character
$< P_+ >$	One primary target or more
$< S_+ >$	One secondary target or more

Table 5.1: *Different configurations of characters*

A camera behavior is finally constructed by manually mapping a set of PSL shot specifications with a set configurations of characters (one specification per configuration). The configuration represents the stimulus of the camera, and the PSL specification represents the response to the stimulus by the camera. For example, an over-the-shoulder behavior (a point of view always behind the shoulder of the character) can be specified on the player character as illustrated in Table 5.2. Our system requires a mandatory specification whenever no configuration can be matched (the default case provided in the Table 5.2).

Behaviors can then be attached to as many cameras as desired in the scene. It is then the role of the Director to select the appropriate rushes from the different cameras.

Configuration	**Specification**
$Default$	MCU on PC 3/4 backright screenleft
$< PC, P_0 >$	CU on PC 3/4 backright screenleft and P_0 screencenter
$< PC, P_+ >$	CU on PC 3/4 backright screenleft and P_+ screencenter

Table 5.2: *A camera behavior encoding the Over-the-shoulder principle on the player character (PC).*

5.4.3 Editing

Once the rushes are generated by the different cameras, the director takes care of editing the rushes to output the final cinematic replay. We perform the editing using the framework presented in chapter 4. With a default average shot length of five seconds the optimization process automatically computes an edit, hence providing the final replay. The importance used as input to the system is based on I_{atomic} and is computed as follows:

$$I(b,t) = \sum_{a \in \mathcal{A}_{c,t}} R(a,t) \times S_c(a,t) \times I(b|a)$$

Where $I(a|b)$ is the importance of a body part b in action a. It is obtained by casting actions into one the four categories already described in the previous chapter.

5.5 The Cinematographer: from specifications to camera coordinates

The purpose of the Cinematographer component is to translate a given PSL specification into camera coordinates: position, orientation and focal length for each frame of the animation.

The automated computation of camera coordinates given a PSL specification is not straightforward. The problem is strongly under-constrained – there are many camera coordinates satisfying the same specification – and the space of possibilities is continuous in the 7D space. In related contributions, more specific camera description languages have been proposed ([OHPL99, BZRL98] and [RU14]). Sophisticated optimization techniques were proposed to compute camera coordinates by expressing properties of the camera with an aggregated cost function (genetic algorithms in [OHPL99], gradient descent [DZ94], particule swarm optimization [RU14]).

However, the technique proposed by [LC12] provides the algebraic solution to efficiently solve a subset of camera properties (exact screen location of two targets, distance to targets, viewing angle). We propose to extend this technique for two purposes: (i) to efficiently compute camera coordinates satisfying a PSL specification, and (ii) to smoothly animate the camera while the geometry is changing or to enable smooth transitions between different PSL specifications.

5.5.1 Computing camera coordinates

We propose to express the computation of camera coordinates from a PSL specification using an incremental pruning process that will successively prune regions of the search space. The search space is define by either a spherical surface or a toric surface [LC12] depending on whether one or two targets are specified. The incremental pruning is performed until all properties of the PSL specification are satisfied, or until an inconsistency is encountered (see Figure 5.4). The output of the process is a region of a surface in which all points satisfy the PSL specification.

Before detailing the pruning process, we propose a new solution to address the singularity observed with the toric manifold in chapter 3.3 when the camera gets too close to one of the visual targets. As illustrated in Figure 5.5, when the camera is too close to points A and B on the toric surface, and when considering that A and B are complex objects such as virtual

characters, the corresponding viewpoint will be of bad quality (having a camera either inside one of the objects, or too close to an object to create a good viewpoint).

We propose in a first stage to extend this toric surface by introducing a threshold value d preventing the camera from being too close to targets A or B. This occurs at the cost of losing the exact composition of A and B in these regions, but improves the quality of the viewpoint. This solution is an improvement over the Bezier curve method that we proposed in chapter 3.

We use this threshold value d between the camera and the targets to alter the surface of the manifold in the following way. For a given arc-circle of the toric (*i.e.* a given value of the vertical angle φ on the parametric surface of the toric), we compute the intersection point I between the arc-circle and the circle defined by either A or B and radius d. The curve is then replaced by the arc circle of center C_2 and radius $|C_2I|$ (in blue on Figure 5.5) where C_2 is the intersection between (C_1A) and (AB). The arc circle provides a C_1 continuity with the initial curve, hence creating a smooth transition that will be useful when animating the camera (see Section 5.5.2).

For a PSL specification with two targets A and B (actors or objects), each with their optional parameters (*e.g.* <*screen*>, <*size*>, <*angle*>, <*profile*> and *(back|fore) ground*), we then apply an incremental pruning process on the extended toric surface by considering the following stages:

1. construct the extended toric surface for the two targets defined by $\langle FlatComposition \rangle$. If no $\langle screen \rangle$ specification is provided, a default one is proposed (first target on the left, second on the right),

2. use the $\langle size \rangle$ specification to compute a vertical band on the toric surface (*i.e.* pruning values of θ). The band represents a range of distances corresponding to the specified shot size. Knowing the camera focal length and the size of the target, the Medium-closeup specification is straightforwardly expressed as a distance δ to the target to which we add some flexibility $(\pm\epsilon)$, then converted to a range values for θ – see [LC12] for more details and illustration in Figure 5.4(a).

3. use the $\langle angle \rangle$ specification to compute a horizontal band on the toric surface (*i.e.* pruning values of φ see Figure 5.4(b)),

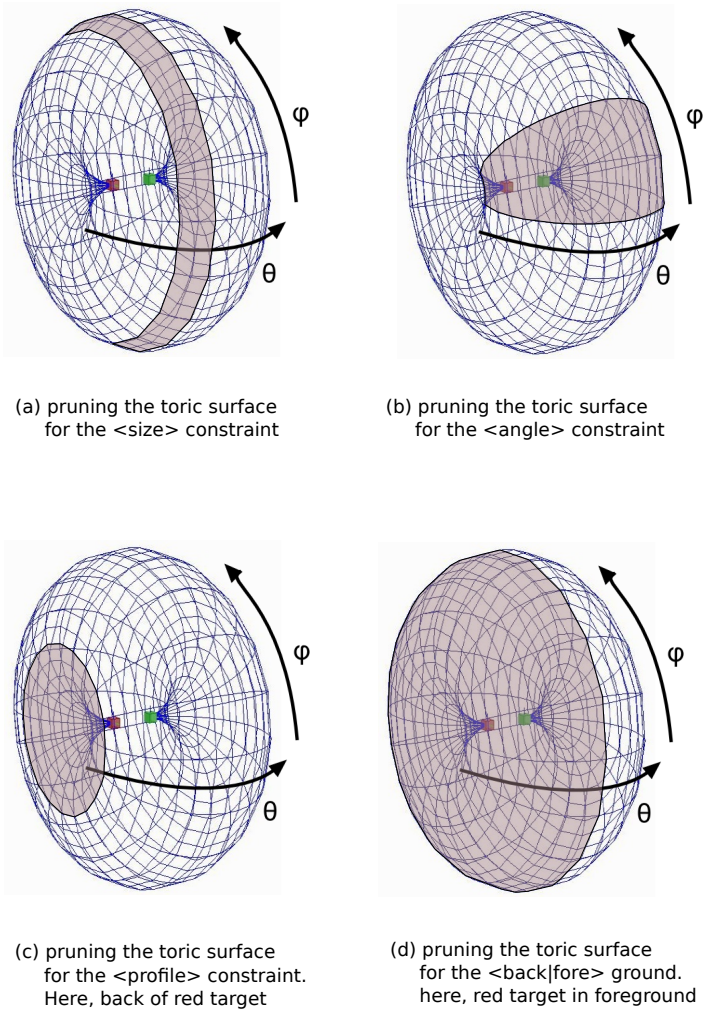
4. use the $\langle profile \rangle$ specification on each target to prune values of θ, by computing the intersection between the specified wedge of the target (*e.g.* 3/4 left back) and the toric surface (see Figure 5.4(c))

5. finally use the *(back|fore) ground* specification to decide whether the camera is closer to actor A or to actor B (hence pruning values of θ, see Figure 5.4(d)).

Each pruning process is performed on the result of the previous stage. Given that some specifications may be optional, not all the pruning stages are performed. At each stage, the pruning process may lead to an empty set corresponding to an inconsistent specification. In such cases of impossible PSL querries, a failure message is returned.

For a PSL specification with only one target, the same pruning process is applied on a spherical surface, using spherical coordinates φ, θ, r. In such case, $\langle size \rangle$ defines a range of values for the radius r. For a PSL specification with more than two targets, the toric surface is constructed using the pair of targets having the greatest distance between them.

Using the spherical or toric surface models, our technique efficiently computes ranges of parameters using the PSL specification. By selecting any given value in these ranges, one can compute a precise camera location and orientation satisfying the specification.



(a) pruning the toric surface for the <size> constraint

(b) pruning the toric surface for the <angle> constraint

(c) pruning the toric surface for the <profile> constraint. Here, back of red target

(d) pruning the toric surface for the <back|fore> ground. here, red target in foreground

Figure 5.4: *The pruning process applied on the toric surface to satisfy terms of the PSL shot specification for two targets A and B (displayed in red and green). The intersection of the regions represent the locations in which to position the camera.*

5.5.2 Animating cameras

The next issue consists in providing means to smoothly animate the camera in two different contexts: (i) maintaining a PSL specification while the scene geometry is evolving, typically when the targets are moving, and (ii) performing transitions between different PSL specifications (either due to a change in the target list, or to a failure in computing a shot satisfying a PSL specification).

Drawing our inspiration from the model we detailed in chapter 3, we propose a physically-based camera animation model that relies on forces directly driven by our extended toric surface or spherical surface and constrained by the 3D environment. The model considers the camera as an oriented particle, influenced by forces guiding both its position and orientation (i) towards

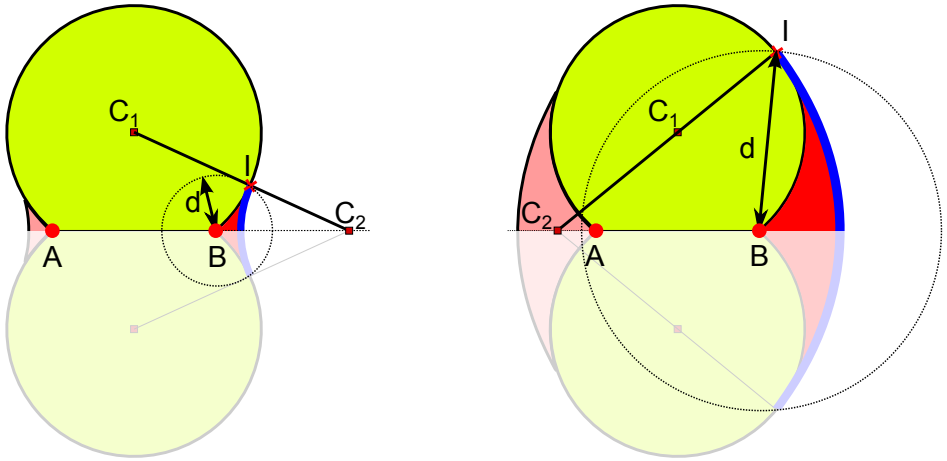


Figure 5.5: *Modified toric surface. The toric surface is modified for camera positions closer than threshold value d from either target A or B so as to avoid collisions. The process is illustrated for a given value φ of the toric surface, and two different threshold values d. The modified region is replaced by the blue arc circle of center C_2, and radius $|C_2I|$ where I is the intersection of the circle of center B and radius d, and C_2 is the intersection of lines $(C_1)I$ and (AB).*

the appropriate range of viewpoints satisfying a PSL specification (the positioning force) and (ii) avoiding collisions with obstacles in the environment (the containment force).

The positioning force helps to maintain a consistent framing of the targets by ensuring the continuous satisfaction of a PSL specification. The force is expressed with two distinct forces: one that pushes the camera on the spherical or toric surface, and another force that pushes the camera on the surface until it reaches a desired position. Algorithm 8 details the computation of these two forces. Figure 5.6 illustrates the idea behind this force using our modified toric surface: we compute the projection P of the camera on the surface and steer the camera to this position (force $\overrightarrow{F_1}$) while pushing the camera on the right hand side or the left hand side, towards the desired position D (force $\overrightarrow{F_2}$). The camera is steered on the right when the desired position D is on the right side of the vector going from the camera C_i to the point C (middle of the two targets). The camera C_2 illustrates the reason for which we don't simply steer the camera directly towards the desired position: with a camera following the red line, the composition will not be ensured, and the resulting viewpoints would be of low quality (having the camera between the targets).

The containment force maintains the camera away from obstacles in the environment (typically walls). Figure 5.7 illustrates the computation of the force and algorithm 9 details its implementation.

The key benefit of this physical camera animation system is to generate smooth camera motions, and to provide control over the transitions between different PSL specifications.

5.5.3 Filtering

Using a physically based model to control cameras offers a practical way to avoid unrealistic camera movements and ensures continuity. The current solution however comes with a drawback: since the toric and spherical surfaces are directly computed from targets' positions, any noisy motions in these positions (nodding, head motion due to walking) will directly impact

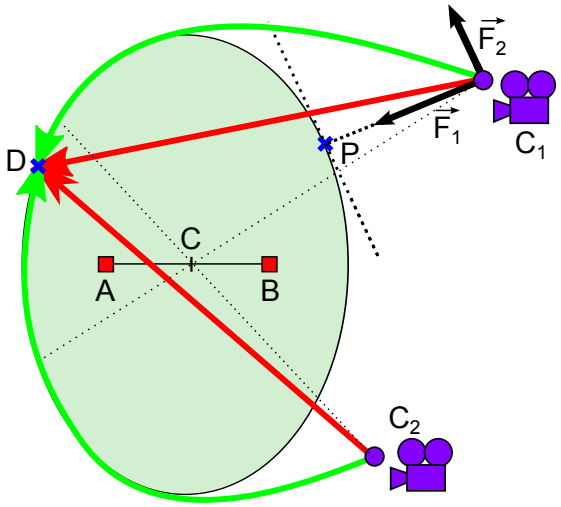


Figure 5.6: *Steering the camera towards the toric surface (force F_1) and steering the camera along the surface towards target D (force F_2).*

Algorithm 8 Positioning: computes the two forces $F_{projection}$ and $F_{targeting}$ which push the camera towards the desired position while staying on the toric surface. P is the projection of the camera position C_i of camera agent i at time t on the manifold surface and D its desired position. $right$ represents the tangent vector of the manifold surface at P. And v_{max} is the maximum allowed velocity for the camera. v_c represents the current camera velocity.

$F_1 = arrive(P)$
$aim = D - P$
// move the camera to the left or to the right
if desired position on the right **then**
  $dir = right$ *// compute a desired velocity to the left*
else
  $dir = -right$ *// compute a desired velocity to the right*
end if
$u = v_{max}((aim \cdot dir)dir + (aim \cdot up)up)$
// subtract the current velocity to the desired velocity
$F_2 = u - v_c$
$F_{framing} = F_1 + F_2$

the camera motions. Figure 5.8 illustrates these issues on target trajectories. Even though the use of a physical system can dampen some of these noisy motions, a more elaborate model is necessary.

While a simple solution could be to apply thresholds to the camera forces to prevent moving the camera when unnecessary, it requires a lot of parameter tuning, induces undesirable motion such as peaks in the acceleration and leads to latency in camera tracking.

To solve this problem, we cast it into a denoising problem by considering the small variations in the target trajectories as noise. The filtered trajectories are then obtained using a total variation (*TV*) regularization algorithm. The idea of using *TV* algorithm for denoising was introduced by [ROF92]. Their use in cinematography was introduced by [GKE11] for

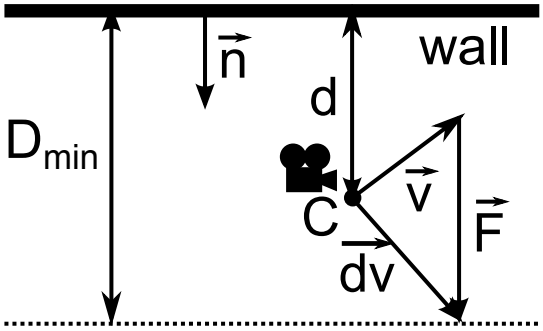
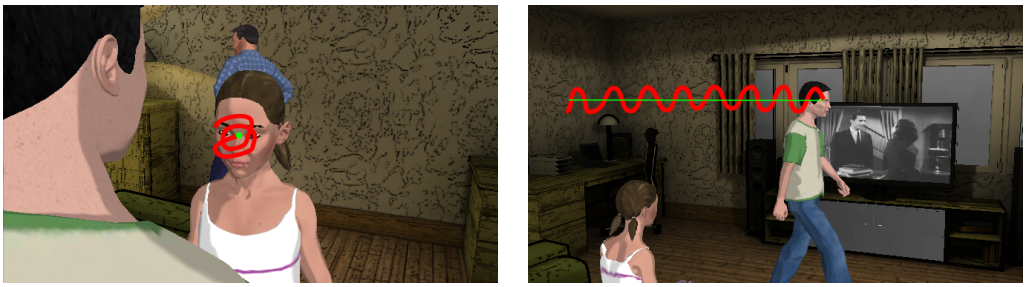


Figure 5.7: *Obstacle avoidance: compute the force F that pushes the camera C away from an obstacle. D_{min} represents the threshold distance, n is the normal of the surface at the closest distance from the camera to the obstacle, v_c is the velocity of the camera and dv the desired velocity*

Algorithm 9 Containment: computes a sum of forces F_{obs} that pushes the camera away from the obstacles. l_i represents the normalized look at vector (orientation) of camera particule i at time t, r_i represents the normalized right vector of camera particule i at time t and v_{max} is the maximum allowed velocity for the camera. D_{min} represents the distance threshold under which the force is applied.

for each obstacle o **do**
  $d = distanceToObstacle(o, p_i)$
  // check whether the wall is under a threshold distance
  if $d < D_{min}$ **then**
    // compute the magnitude of the force
    $mag = D_{min} - (d + (v \cdot n))$
    $F_{obs} = F_{obs} + n * mag$
  end if
end for



(a) *Continuous head movement* **(b)** *Oscillations of the character's head*

Figure 5.8: *A denoising algorithm is applied on the motion of the character (eg balancing head motions, or head walking motions) to prevent noisy camera trajectories.*

video stabilization. The idea behind the *TV* denoising problem is the following: we are given a (noisy) signal $y = (y[1], ..., y[N]) \in \mathbb{R}^N$ of size $N \geq 1$, and we want to efficiently compute

the denoised signal $x* \in \mathbb{R}^N$, defined implicitly as the solution to the following minimization problem with a regularization parameter $\lambda \geq 0$:

$$\underset{x \in \mathbb{R}^N}{minimize} \frac{1}{2} \sum_{k=1}^{N} \left| y[k] - x[k] \right|^2 + \lambda \sum_{k=1}^{N-1} \left| x[k+1] - x[k] \right|$$

For the purpose of filtering trajectories, the denoising is performed by applying the *TV* regularization to each of the coordinates (*x*, *y* and *z*) over time (*N* thus represents the number of frames of the sequence). To obtain smooth and steady camera movements, we propose to denoise the target's trajectories as a pre-process (rather than denoise the computed camera motions). We keep the advantage of the force-based system by tracking trajectories that have already been filtered and thus do not induce extra forces to constantly adjust the camera when it is not needed.

For denoising the trajectories, we used a direct non-iterative algorithm presented by [Con13]. Finding the appropriate value for parameter λ was performed through multiple experimentations. The value was finally set to *2.0*.

5.6 Experimental results

To demonstrate our approach, we used the video game *Nothing For Dinner*. This interactive drama presented in [HSRD12] and available online[2] uses the story engine IDtension. The goal of this serious game is to help teenagers cope when a parent suffers from traumatic brain injury. The simulation immerses the players in an interactive environment in which they play active roles and have to make decisions that require their attention. The gaming experience provided by *Nothing For Dinner* gives users a way to experience different situations that they might encounter in their everyday life. We integrated our cinematic replay system within this serious game, giving the possibility for users to replay their experiences.

5.6.1 Narrative importance

All the narrative information is generated by the IDtension engine and saved for further analysis by our system. What is being generated by IDtension could be considered as the fabula of the story: it contains all events and actions occurring during the game session along with their temporal relations within the fictional world without any consideration of viewpoint or focalisation. To generate the cinematic replay, our system extracts information from this fabula, typically beats and atomic actions. Each atomic action is described with the following attributes: *starting time*, *duration*, *type of actions* and *description*.

The information on the relevance of the actions performed by the characters is part of the internal mechanisms of *IDTension*. It is termed *motivation* and corresponds to the relevance of the action in terms of the accomplishment of the character's goal. Combined with the significance of character's role in each action, this metric provides a means to establish the individual importances of the characters.

5.6.2 Shots specifications

For the results, we demonstrate the capacities of our system by using only 3 cameras. Two cameras rely on the fine-grain importance I_{atomic} and the third one (the master shot) relies on

[2] http://tecfalabs.unige.ch/tbisim/portal/

the beat importance I_{beat}. Tables 5.3, 5.4 and 5.5 describe the behaviors defined for each of these cameras. With this implementation, the first camera represents the Point-Of-View shot from the player character's perspective and the second camera represents its reverse shot.

Configuration	**Specification**
$Default$	MCU on PC 3/4 backright screenleft
$< P_0 >$	POV PC on P_0 screencenter
$< PC, P_0 >$	POV PC on P_0 screencenter
$< PC, P_+ >$	POV PC on P_+ screencenter

Table 5.3: *Behavior for the first camera*

Configuration	**Specification**
$Default$	MCU on PC 3/4 right screencenter
$< P_0 >$	CU on PC 3/4 right screencenter
$< PC, P_0 >$	CU on PC 3/4 right screencenter
$< PC, P_+ >$	CU on PC 3/4 right screencenter

Table 5.4: *Behavior for the second camera*

Configuration	**Specification**
$Default$	MS on PC right screenleft
$< P_0 >$	MS on PC screenleft, P_0
$< PC, P_0 >$	MS on PC screenleft, P_0
$< PC, P_+ >$	MS on PC screenleft, P_+

Table 5.5: *Behavior corresponding to a master shot*

5.6.3 Computing camera positions

To evaluate the cinematography, we present qualitative results produced by our system. Figures 5.9, 5.10 and 5.11 show shots generated for different situations using different camera behaviors. Figure 5.9 shows the output of the three cameras when no specific action is occurring. The camera simply performs a tracking of the player character PC. Figure 5.10 shows the results obtained in a situation of dialog between the player character and another character. Figure 5.10a shows the Point Of View shot obtained using the set of rules previously defined and Figure 5.10b shows its reverse shot: the internal shot.

To illustrate the benefit of the system, we show how a change in camera behaviors impacts the computed viewpoints. Rather than using a Point Of View shot combined with an internal shot, we used two complementary Over-The Shoulder-shots. To produce the result displayed in Figure 5.11 – to be compared with Figure 5.10, we simply replaced the following rules respectively for the first and second cameras, thereby offering simple means for users to author their cinematic replays without manually moving the camera and re-editing the sequence.

- $< PC, P_0 >$: CU on PC 3/4backright screenleft and P_0 center
- $< PC, P_0 >$: CU on P_0 3/4backleft screenright and PC center

(a) *First camera* **(b)** *Second camera* **(c)** *Master shot*

Figure 5.9: *Shots computed for three different camera behaviors on the same scene at the same time (a) first camera behavior, (b) second camera behavior and (c) master shot behavior.*

(a) *First camera* **(b)** *Second camera* **(c)** *Master shot*

Figure 5.10: *Shots computed for three different camera behaviors on the same scene at the same time: (a) point-of-view behavior defined on the PC Frank, (b) point-of-view behavior defined on P_0 Lili and (c) master shot behavior defined on $< PC, P_0 >$.*

(a) *First camera* **(b)** *Second camera* **(c)** *Third camera*

Figure 5.11: *Shots computed for three different camera behaviors on the same scene at the same time: (a) over-the-shoulder behavior defined on Frank, (b) over-the-shoulder behavior defined on Lili and (c) master shot behavior on Franck and Lili.*

5.6.4 Overall process and results

These few examples illustrate the type of camera shots generated by our system. It highlights the complementarity of the behaviors in generating various shots that makes the editing process easier. To illustrate the overall system, Figure 5.2, already introduced in section 5.2, presents the complete cinematic replay process. The process starts with the analysis of the list of actions and activities to compute both the atomic and beat importances of the character along the timeline. The figure shows the evolution of the atomic importance of the characters over time (the same computation is performed for the beat importance). Using this information at each time step we can extract the list of characters involved (configurations of characters) and use it to define the camera specifications from the set of behaviors presented in Tables 5.9, 5.10 and 5.11 (in this case, Frank is the Player Character). The rushes are then computed from the

camera specifications using the steering behaviors and the editing between the different rushes is performed.

Finally, a video demonstrating the results can be seen online[3]. It presents two different replays of the same game session. We generated them by changing the behaviors of the camera as mentioned before. This video shows that a small set with three cameras and only a few rules is enough to cover basic interactions between characters and transitions between actions.

5.7 Limitations and Future Work

In this chapter, the focus was set on the generation of cinematic replays for dialogue-based role playing games. It provides a generic solution for this purpose but doesn't make full use of the narrative information that some games or interactive narratives might provide. Looking at richer information, the proposed cinematography system could be improved, for example by addressing the emotion of the characters. Though the game *Nothing For Dinner* itself provides us with such information, the automated computation of compelling cinematographic sequences conveying emotions remains an open challenge. Future work could be pursued based on the studies conducted to evaluate the impact of camera control on the emotional state of the audience [MJY09, YMJ10] – even though these studies were related to the gaming experience, their results could be extended to cinematographic purposes.

In addition, the method presented for the computation of camera shots from PSL specifications proved to be efficient on simple cases but does not handle over-constrained situations. When the system can not find a solution, it immediately falls back to a default configuration with minimal constraints and guaranteed solutions. Investigation of constraint relaxation methods could be conducted to properly address this problem and provide approximating solutions for inconsistent specifications.

Finally, the steering behaviors method used for the cinematography remains a reactive approach. As such, combined with an on-line camera editing technique (such as [CAH*96]), it could be adapted to address real-time contexts. However, when real-time is not a constraint, such a method does not fully benefits from the off-line advantages. For instance it can not use information on characters' trajectories to anticipate the movement and globally improve the camera path. Offline optimization techniques are more suited for this task. Moreover, even though this technique constrains cameras in terms of speed and acceleration to ensure realism, it still allows free displacements in the environment which sometimes results in unrealistic trajectories. In chapter 6, we address both these limitations and make full use of the off-line property of the system to ensure plausible camera trajectories and motions by using virtual camera rails.

5.8 Summary

In this chapter we have presented a new system designed to automatically generate cinematic replays of game sessions. We presented a new way to define high-level camera specifications using a principled and contextual importance-driven approach, as an answer to the limitations of action-based or idioms-based cinematography. We also introduced a mean to express camera behaviors using these specifications, and proposed novel techniques to smoothly animate the cameras. The results obtained with only three camera behaviors illustrate the capacity of the system to properly convey a game replay, with a realistic and dynamic camera style.

[3] https://team.inria.fr/imagine/narrative-driven-camera-control/

CHAPTER

6

CAMERA-ON-RAILS

THough our previous work on camera control gave satisfying results to automatically compute camera paths in virtual environments, it does not seriously consider the problem of generating realistic camera motions even for simple scenes. Among possible cinematographic devices, real cinematographers often rely on camera rails to create smooth camera motions which viewers are familiar with.

Following this practice, in this chapter we propose a method for generating virtual camera rails and computing smooth camera motions on these rails. Our technique analyzes characters motion and user-defined framing properties to compute rough camera motions which are further refined using constrained-optimization techniques. Comparisons with recent techniques demonstrate the benefits of our approach and open interesting perspectives in terms of creative support tools for animators and cinematographers.

6.1 INTRODUCTION

Computing smooth and well-composed camera paths is a key problem in both real and virtual cinematography. In both contexts, a crucial step is to provide cinematographers with means to design camera motions which, on one side enable to track the motion of targets (often characters) while on the other side maintain the camera motions as smooth as possible – this is particularly desired in real cinema, where minimizing the optical flow is one of the main concerns.

In related work, virtual cameras have often been considered either as able to move completely freely, or been restricted to simple stereotypical motions (*e.g.* traveling, arcing, dolly-in). Conversely, in the real cinema industry, camera motions are often restricted to those of a camera rig (*e.g.* a Louma or a dolly) moving along a single continuous rail. Generating realistic camera motions in such a context remains a tedious task that requires iterating on (i) positioning the rail in the scene, then (ii) generating a camera motion along this rail (through a fine control of the on-screen layout and camera speed), and finally (iii) viewing the on-screen result to decide whether or not changes should be operated on the rail positions or on the camera motions.

In an attempt to reproduce cinematographic practices to create realistic and smooth camera paths, we propose a two-stage approach to the automated creation of camera paths. In the first step, we compute a virtual camera rail that will constrain the camera motions while the camera is transitioning between two framings. In the second step, we generate a smooth camera motion onto this rail, that ensures a proper composition of target characters on the screen.

Ranging from the tracking of simple motions of characters to the creation of camera movements portraying complex motions of characters, our system provides means to efficiently compute stereotypical and natural camera shots through the simple specification of initial and final framings. As a result, our method allows to automatically create rushes that can be used either as input to an editing or previsualisation system (such as the one presented in chapter 4 or the work by Lino *et al.* [LCRB11a]), or to create a single extended shot (long take) which conveys a whole movie scene.

The contributions detailed in this chapter are:

1. a method to compute virtual camera rails that enable creating realistic camera transitions between an initial and a final framing, while accounting for the overall motion of target characters;

2. a method to compute smooth camera paths constrained on a given rail. Our method nicely combines constraint solving and optimization to account for both aesthetic constraints (*e.g.* frame characters all along the camera path) and constraints on the camera (*e.g.* smoothly adapt the camera velocity and acceleration, and prevent the camera to get too close or too far from characters).

The chapter is built as follows. We first present an overview of our method (Section 6.2). We then detail our two contributions (Sections 6.3 and 6.4). We further compare and discuss our solution with other methods and demonstrate its use in several context(Section 6.5). Finally, we conclude and present perspectives for future research (Section 6.7).

6.2 OVERVIEW

In this chapter, we focus on the automated computation of natural camera motions between an initial and a final framing specified on a single character or on two characters. A framing

in this context represents the specification of the visual layout of characters on the screen in terms of exact on-screen positions, relative camera angle (high to low, front to back [Ari76]) or on-screen sizes.

As in the real cinema industry, our camera planning method is divided into two consecutive steps: (i) creating a camera rail, and (ii) moving the camera along the rail. The first step consists in computing a camera rail that links an initial and a final framing, and along which the characters' motions can be viewed. The second step then consists in computing appropriate camera orientations and speed along the rail so as to track the characters. To this end, we rely on a constrained-optimization process which accounts for the framing of characters, as well as the speed and acceleration of the camera along the camera rail.

6.3 BUILDING CAMERA RAILS

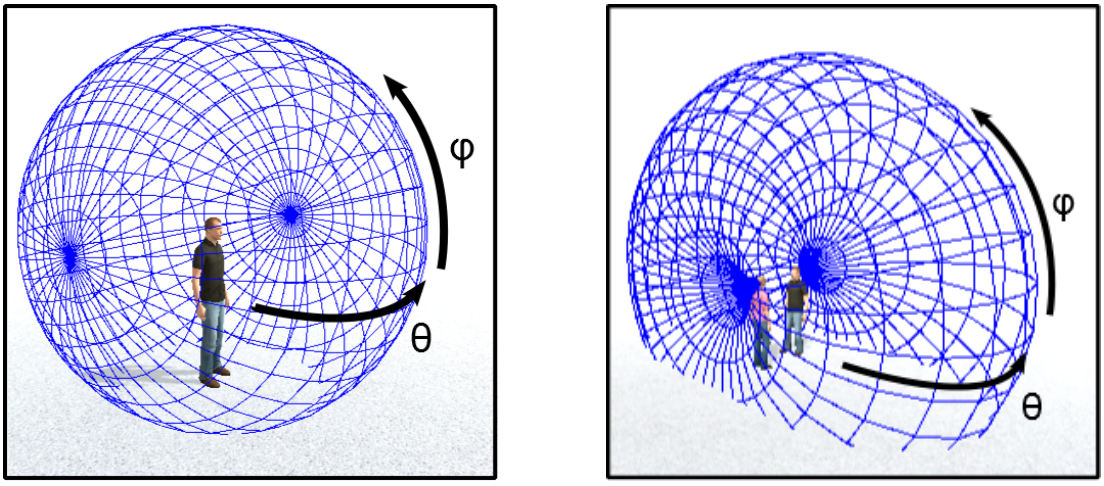
In order to build a camera rail, we first rely on a parametric representation of viewpoints similar to the one proposed in [LC12]. This allows us to create a *raw trajectory* which satisfies, at each frame, the exact linearly interpolated framing between the initial framing and the final framing. This raw trajectory generally results in jerky camera motions. We then approximate the *raw trajectory* with a Bezier curve so as to smooth it out. We restrict the complexity of the camera rail to a cubical curve, so as to limit rails to those that are commonly used in real movies, hence enhancing the naturalness of created shots.

6.3.1 Computing a raw trajectory

The input of this process is the motion of the characters as well as the user-specified initial and final framings –at times t_0 and t_1 respectively– of these characters. These framings comprise on-screen desired positions, on-screen sizes and vantage angles. Different optimization techniques can be used to compute actual camera configurations from these framing specifications; we here rely on [LC15] that provides an efficient and algebraic implementation.

Initial and final camera configurations can therefore be expressed into a 2D-parametric representation, using the manifold surfaces mentioned in the previous chapter: a spherical surface (that can handle single-character configurations) or a toric-shaped surface (that can handle two-character configurations). In the case of a single character, the sphere is defined by using the shot size of the character. The horizontal and vertical vantage angles are defined in spherical coordinates, as shown in Figure 6.1a. The on-screen position of the character is finally determined through its projection onto the image plane. In the case of a pair of characters, we rely on the toric-shaped manifold surface proposed by Lino and Christie [LC12] to compute the camera settings from the viewpoint. As explained by Lino and Christie, the toric surface is fully determined by the on-screen positions of the pair of characters. The vantage angles are then computed in a way similar to the single-character case, as shown in Figure 6.1b.

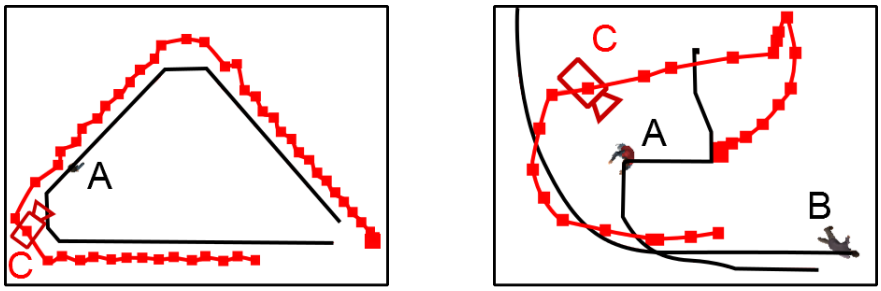
To compute the *raw trajectory* linking the initial and final camera configurations, we propose to interpolate their framing properties (on-screen position, view angle and size) along time, and compute for each time step, the camera configuration satisfying the interpolated framing. Framing properties such as characters' on-screen positions, vantage angles or sizes are computed through a straightforward linear interpolation. In the case of a pair of characters, it is however required to distinguish two types of viewpoint interpolations. Indeed, since the vantage angle and the size of the target characters are correlated (getting closer to one character changes the vantage angle on the other), it is only possible to constrain (and thus interpolate)



(a) *Spherical surface* **(b)** *Toric surface*

Figure 6.1: *Manifold surfaces used to defined visual properties in the case of (a) one target and (b) two targets. In both cases, profile and vantage angles are given respectively by θ and φ.*

one of them. Figure 6.2 illustrates the computation of the raw path which interpolates between both viewpoints.



(a) **(b)**

Figure 6.2: *Example of raw trajectories computed in the cases of (a) a single moving target A and (b) two moving targets A and B. The camera raw trajectory is displayed in red while the targets' trajectories are displayed in black.*

6.3.2 From raw trajectories to camera rails

The computation of a camera rail then requires a curve fitting process to create a rail as close as possible to the raw trajectory (hence satisfying the framing properties along the path), while smoothing it out. While a classical B-spline model provides a good fitting, it may result in the creation of complex, hence unnatural, camera paths. Simpler models such as quadratic Bezier curves (as suggested by [OHPL99] in their CAMPLAN system) remain limited (*e.g.* cannot handle loops). We here propose a cubic Bezier curve, which offers a simple representation of a camera rail with limited curvature changes and yet provides sufficient flexibility to handle most common camera motions from the literature [Ari76].

A rail R is therefore defined by a parametric function with four control points (P_0 to P_3). P_0 and P_3 represent the extremities of the rail, while P_1 and P_2 represent tangents at these extremities (hence controlling the curvature and shape of the rail). Any point on R can be computed from parameter $t \in [0, 1]$ using the equation

$$R(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + P_3t^3$$

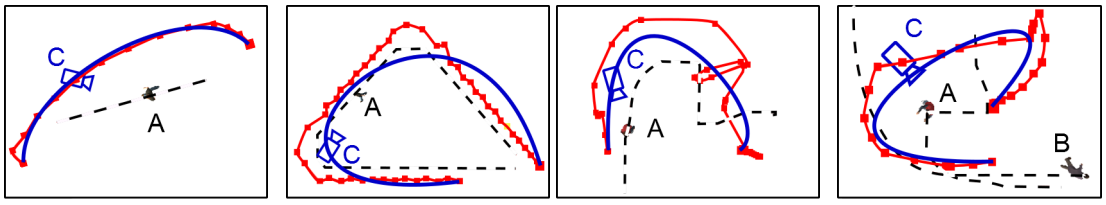
To approximate the raw path (which we will refer to as a dataset D), we compute the Bezier parameters by using a least squares fitting method. At each camera position D_i along the raw path, we first associate a parameter t_i computed as the cumulative distance (following the raw path) to reach D_i, divided by the total length of the raw path. We then minimize an error ξ defined as the cumulative squared distance between all camera positions of the raw path and their corresponding positions onto the Bezier curve R. This error is computed as

$$\xi = \sum_{i=0}^{N} (R(t_i) - D_i)^2$$

Since our rail needs to operate the exact transition between the initial and final viewpoints, we can directly assign the first and last control points to the initial and final camera positions of the raw path respectively. At this point, only the positions of P_1 and P_2 are left unknown. To find the minimum value of the error function, we compute partial derivatives with respect to these two unknowns and find where these equal zero, *i.e.*

$$\frac{\delta\xi}{\delta P_1} = 0 \text{ and } \frac{\delta\xi}{\delta P_2} = 0$$

Finally, as the maximum error is infinite, we know that the solution of the system corresponds to the minimum error. To compute control points P_1 and P_2, we then solve this system of linear equations. Figure 6.3 shows how our method approximates complex raw paths with simple camera rails.



(a) **(b)** **(c)** **(d)**

Figure 6.3: *Examples of camera rails (in blue) computed to approximate raw trajectories (in red) when tracking one or several characters with increasingly complex motions (from (a) to (d)). As the motion of the character becomes more complex, our method still provides a rail that well approximates the raw path.*

6.4 MOVING THE CAMERA ON THE RAIL

We now focus on generating a smooth camera motion (in position and orientation) along this rail while maintaining an optimal framing over the target characters. To address this problem, we first compute at each time step t_i, the optimal camera position on the rail that satisfies the

interpolated framing at t_i. We then improve this camera motion through an iterative optimization process which accounts for constraints on the camera (in terms of position, velocity and acceleration). Hence, we compute a camera motion closest to the optimal framing while enforcing smoothing constraints. We perform a similar process to compute the orientation of the camera along the path. These stages are described in the following sections.

6.4.1 Raw camera motion

To initiate a raw camera motion along the rail, we try at each time step to position the camera at an optimal position *on the rail*, knowing it's previously defined raw position, *i.e. on the raw trajectory*. Though a straightforward solution could consist in projecting this raw camera position onto the rail, it may not ensure the satisfaction of visual properties (see Figure 6.4 for an example). Our solution consists in finding the position on the rail that is closest to the intersection between the rail and the manifold surface. Since every position on this manifold surface satisfies part of the desired visual composition (at least the on-screen position of target characters, as shown in [LC12]), this methods tends to provide better camera positions to frame the target characters (and limits disturbing changes in on-screen sizes of characters).

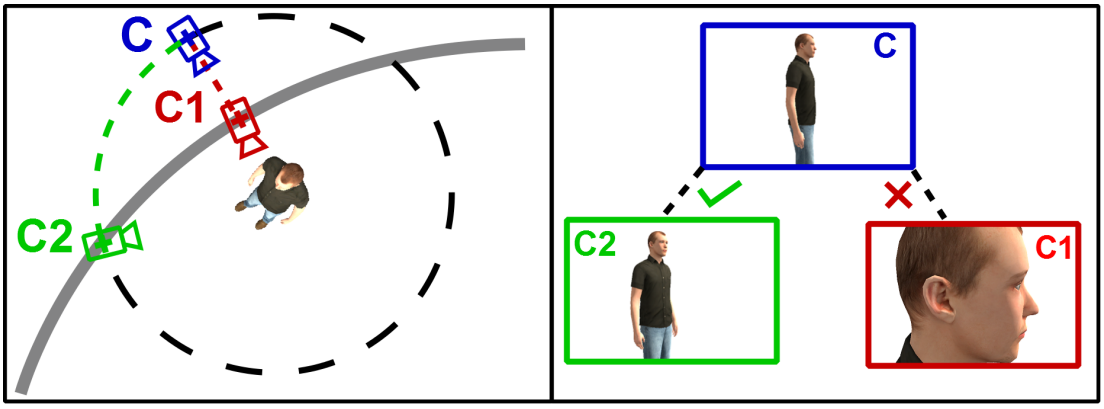


Figure 6.4: *The point on the rail (in grey) that provides the best approximation of the desired camera viewpoint C is not its projection $C1$ on the rail. It is given by the closest intersection $C2$ of the rail with the manifold surface (in black).*

As the intersection of a Bezier curve with a manifold surface is not straightforward, we compute the optimal camera position at a given frame by performing a dichotomous search on the neighborhood (along the rail) of the position found at previous frame. This search algorithm is detailed in Algorithm 10.

6.4.2 Smooth camera motion

To compute a smooth camera motion from the *raw motion* on the rail, we need to add constraints such as smooth changes of camera velocity and acceleration. We therefore search for the camera motion that satisfies these constraints while minimizing the distance to the raw camera motion.

The problem is solved using an optimization process which takes as input a shot duration d (comprising N frames) and a rail of length L. This process then minimizes equation (1), where x_i and x'_i are the parameters respectively defining the optimal and optimized camera position on the rail. Our optimization process is then subject to a number of constraints, each addressing

Algorithm 10 Dichotomous search of the point on the rail that is the closest to the manifold surface. The search is performed in the neighborhood (defined by Δ) of the previous optimal camera position p. $Rail(t)$ returns the position of the point at time t on the rail.

$left := p - \Delta$
$right := p + \Delta$
while $left - right \geq \epsilon$ **do**
  $middle := (left + right)/2$
  $p_{left} := Rail(middle - \epsilon)$
  $p_{right} := Rail(middle + \epsilon)$
  $d_l := ||projectOnManifold(p_{left}) - p_{left}||$
  $d_r := ||projectOnManifold(p_{right}) - p_{right}||$
  if d_l **then**
    $left := middle$
  else
    $right := middle$
  end if
end while
return $(left + right)/2$

one aspect of the motion. Firstly, through constraint (2) we state that the camera position will always belong to the rail, in-between both extremities of the rail. Secondly, through constraints (3) and (4) we state that the camera initial and final positions will be the extremities of the rail (P_0 and P_3 respectively). Thirdly, through constraints (5) and (6) we define a maximum velocity v_{max} and a maximum acceleration a_{max} for the camera motion. Finally, through constraints (7) and (8) we state that the camera motion will start with a zero speed and end with a zero speed (*i.e.* a complete stop of the camera). In other words, our optimization process is formulated as

Minimize

$$\sum_{i=0}^{N} |x'_i - x_i| \tag{6.1}$$

Subject to

$$0 \leq x'i \leq 1 \tag{6.2}$$
$$x'_0 = 0 \tag{6.3}$$
$$x'_N = 1 \tag{6.4}$$
$$|x'_i - x'_{i-1}| \leq v_{max} * dt/L \qquad \forall i \geq 1 \tag{6.5}$$
$$|2x'_{i-1} - x'_i - x'_{i-2}| \leq a_{max} * dt^2/L \qquad \forall i \geq 2 \tag{6.6}$$
$$x'_1 - x'_0 \leq a_{max} * dt^2/L \tag{6.7}$$
$$x'_N - x'_{N-1} \leq a_{max} * dt^2/L \tag{6.8}$$

Now formalized, the problem can be solved using any existing linear programming library[1]. However, to ensure that our problem has a solution, we further define an implicit constraint on

[1] we used GLPK (GNU Linear Programming Kit)

the minimum input values of v_{max} and a_{max} in the following way. If we use the minimum acceleration satisfying all constraints, the camera will constantly accelerate until it reaches half of the rail length at precisely half of the rush duration, then constantly decelerate until it reaches the end of the rail at precisely the end of the rush (with zero speed). This can be formalized in a simple mathematical way, as

$$a_{max} \geq \frac{4L}{d^2}$$

Using this formula, we then deduce the minimum value of v_{max}. We know that the camera will reach half of the rail at precisely half of the rush duration and at its maximum speed. The camera will also constantly accelerate until it reaches its maximum speed, then will remain constant until half of the rail length. This can be formalized as

$$v_{max} \geq \frac{d.a_{max} - \sqrt{d^2.a_{max}^2 - 4L.a_{max}}}{2}$$

6.4.3 Camera orientation

The two previous steps have computed a smooth camera motion (in terms of position) along the rail. Now computing the optimal camera orientation at each frame, given its position on the rail and a framing to satisfy, is easily addressed in [LC12] through an algebraic formulation of camera orientation. However, for the same reasons as before, we also want to limit the angular speed and acceleration of the camera while it is moving along the rail to avoid jerky camera rotations that may occur. In a way similar to camera position, we therefore perform an optimization process along each of the three axes of the camera (*i.e.* pan, tilt, and roll).

This optimization process takes as input the duration d of the rush (comprising N frames) and is defined as the minimization of equation (9), where θ_i and θ'_i are respectively the optimal and optimized rotation along a given camera axis at frame i. The camera orientation is also subject to a number of constraints, both at its initial and final states and on the way it evolves along time. Firstly, through constraints (10) and (11), we state that the initial and final camera orientations will be equal to the initial and final optimal orientations respectively. Secondly, through constraints (12) and (13), we define a maximum angular velocity $\dot{\theta}_{max}$ and acceleration $\ddot{\theta}_{max}$ for re-orienting the camera. Finally, through constraints (14) and (15), we state that the camera will start the rush and end the rush with a zero angular speed. In other words, this optimization process can be formulated as:

Minimize

$$\sum_{i=0}^{N} |\theta'_i - \theta_i| \tag{6.9}$$

Subject to

$$\theta'_0 = \theta_0 \tag{6.10}$$

$$\theta'_N = \theta_N \tag{6.11}$$

$$|\theta'_i - \theta'_{i-1}| \leq \dot{\theta}_{max} * dt \qquad \forall i \geq 1 \tag{6.12}$$

$$|2\theta'_{i-1} - \theta'_i - \theta'_{i-2}| \leq \ddot{\theta}_{max} * dt^2 \qquad \forall i \geq 2 \tag{6.13}$$

$$|\theta'_1 - \theta'_0| \leq \ddot{\theta}_{max} * dt^2 \tag{6.14}$$

$$|\theta'_N - \theta'_{N-1}| \leq \ddot{\theta}_{max} * dt^2 \tag{6.15}$$

6.5 RESULTS

In this section we analyse the performance of our solution on a number of scenarios and compared our approach with existing camera control techniques. We also illustrate our method by applying it on animated scenes to produce short edited videos. All the results are accessible online[2].

6.5.1 Performance

We here provide an overview of the performance of our method for different character placements and motions. The computational time required (on a Core i7@2.4GHz running Unity 5) to generate a camera rail is given in Table 6.1 for three different scenarios (S1 to S3). In each scenario, the camera makes a transition between two different user-defined framings (see Table 6.2). In the first scenario (S1), the camera tracks a single character as he moves in the environment for 32 seconds. In the second scenario (S2), the camera transitions between two different viewpoints specifications around a pair of static characters during 8 seconds. In the last scenario (S3), the camera tracks during 18 seconds two moving characters walking at different speeds and with different walking directions. Figure 6.5 shows the characters' paths and the camera rails computed for each of these scenarios.

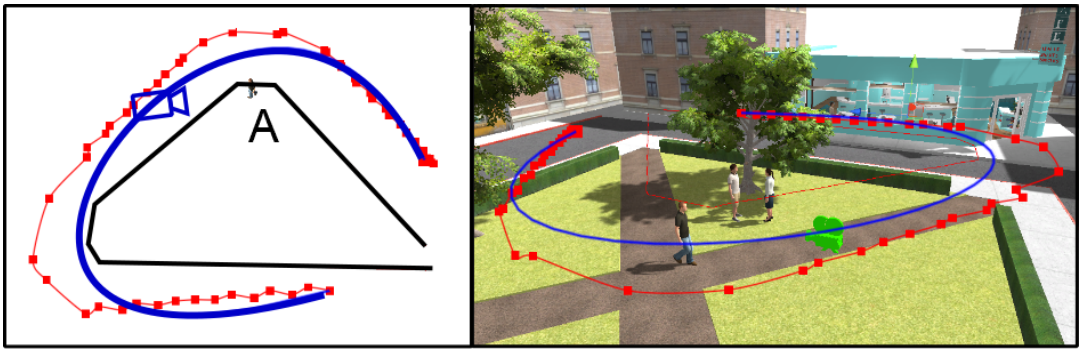
	S1	S2	S3
Raw path	5 ms	13 ms	10 ms
Bezier interpolation	13 ms	14 ms	15 ms
Desired positions on the rail	74 ms	103 ms	342 ms
Optimization of the position	920 ms	49 ms	231 ms
Desired orientation on the rail	7 ms	9 ms	26 ms
Optimization of the orientation	940 ms	117 ms	877 ms
Overall computation	1.95 s	0.30 s	1.5 s

Table 6.1: *Computation times for the different steps of the planning process in three different scenarios (S1 to S3). S1: track a single moving character during 28s. S2: move around a pair of static characters for 8s. S3: track two moving characters for 20s.*

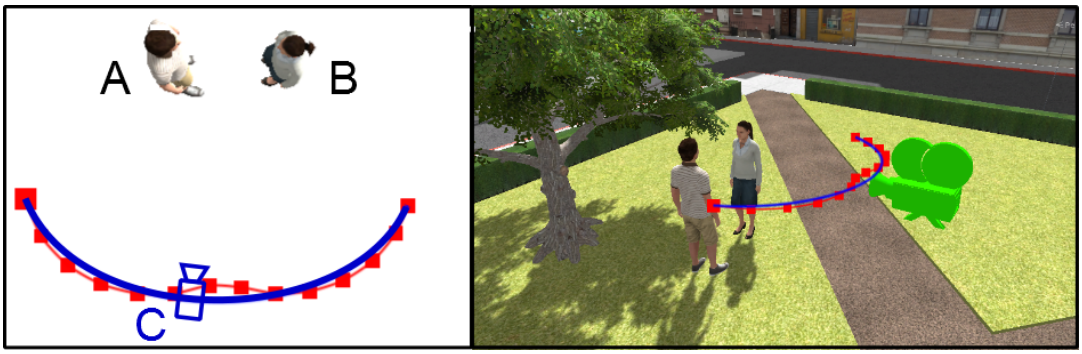
Scenario	Initial specification	Final specification
S1	MFS on P1 3/4 frontright screencenter	MFS on P1 3/4 frontright screencenter
S2	CU on P1 3/4 backleft screenrigh and P2 screencenter	CU on P2 3/4 backright screenleft and P1 screencenter
S3	MS on P1 screenright and P2 screenleft	MS on P1 screenright and P2 screenleft

Table 6.2: *Initial and final PSL specifications used for each scenario.*

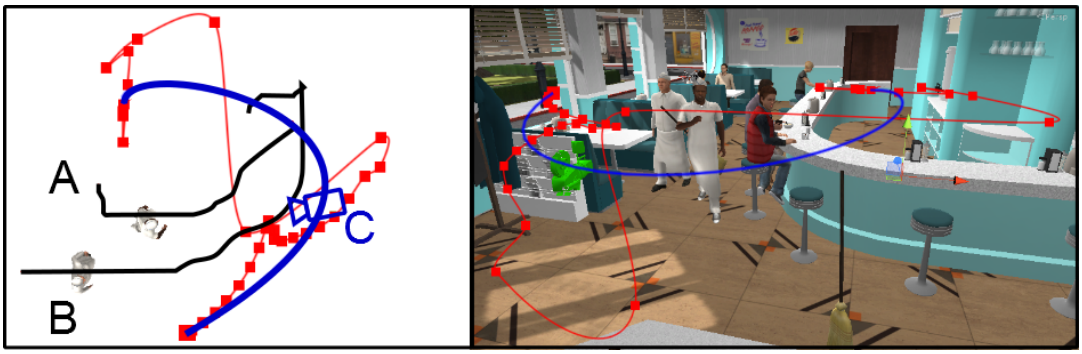
[2] https://team.inria.fr/imagine/camera-on-rails/



(a) *Scenario 1*



(b) *Scenario 2*



(c) *Scenario 3*

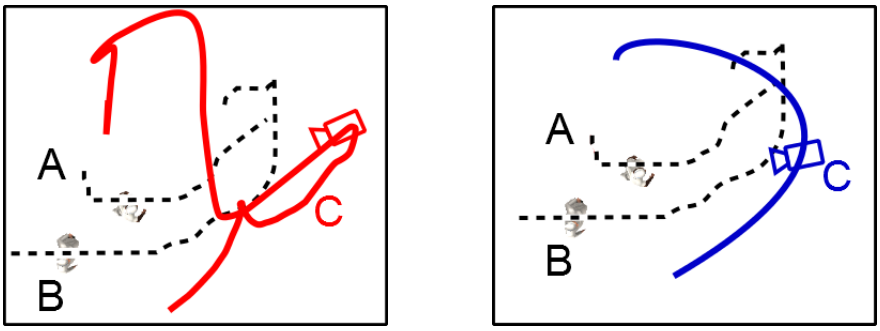
Figure 6.5: *Sample camera rails (blue) computed from the raw camera trajectory (red) for a single moving character (a), for two static characters (b) and for two moving characters (c).*

Table 6.1 shows the computation time spent in each step of the process for each of the three scenarios. The overall computation time to create a smooth camera motion and orientation for the first scenario well illustrates the efficiency of our solution; less than 2 seconds are required to generate a 32-seconds camera motion. Moreover, most of computation time is spent in the optimization process (95% in average). Thus, there is not a significant impact of the number of targets or the type of transition on the overall computational time. We can however notice that the time spent in finding an optimal position on the rail is greater when tracking a pair of targets than when tracking a single character, since working with toric surfaces is more expensive than working with simple spheres.

6.5.2 Comparison with other methods

We compared our method with three other camera planning techniques: (i) an approach relying on the model introduced by Lino and Christie [LC12], (ii) the technique based on steering behaviors presented in chapter 3, and (iii) a derived version of this previous technique where the camera is steered along a virtual rail rather than optimized. All the comparisons are performed on the scenario S3.

Lino and Christie [LC12]. Their technique can be used to create camera motions that strictly enforce a simple framing on a pair of targets. The resulting camera path corresponds to our raw camera path. The main problem of this method is illustrated in Figure 6.6. When the targets' motions are too complex, the method will tend to create jerky and unnatural camera motions. Moreover, as the camera viewpoint is recomputed at each frame without considering either the previous camera position, nor the camera speed, the resulting camera motion is not guaranteed to be continuous nor smooth.

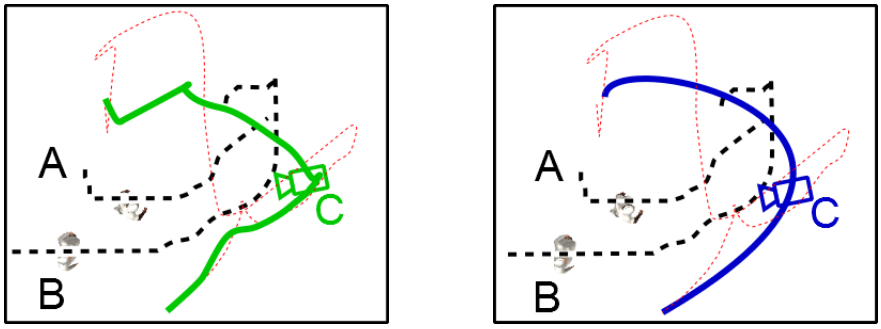


(a) **(b)**

Figure 6.6: *Comparison between the trajectory computed with (a) the Manifold surface introduced by Lino and Christie and (b) our camera rail.*

Steering-based camera control. For this comparison, we use the camera control method presented in chapter 3 and used in chapter 5. An optimal viewpoint is computed and the camera is then steered toward this viewpoint. The resulting camera motions are much smoother than the one obtained with [LC12] and provide interesting results in simple cases. However, as shown in Figure 6.7, when confronted to complex situations, steering-based cameras fail in generating smooth trajectories. Furthermore, the computational time required to compute the camera path from the raw trajectory by using steering behaviors is greater than when using our method. It takes 3.33 seconds to compute both the position and orientation for the whole duration of the rush (given the raw optimal trajectory), whereas when using our optimization process it only takes 1.1 second. Even though our approach only offers an off-line solution – it requires an analysis of the target characters' motions –, it demonstrates better performances in average than when using our real-time method.

Extended version of steering-based cameras. We also compared our solution to an extended version of the method presented in chapter 3. Instead of freely steering the camera around in the environment, we constrain its motion along the same rail as in our method. We then steer, at each frame, the camera towards the optimal viewpoint on the rail by using the steering behaviours of chapter 3. This last comparison is important since it confirms the necessity of the optimization we propose. Indeed, Figure 6.8 shows the evolution of the position of the camera on the rail for both methods and displays the optimal camera position along time. Our method closely follows the optimal position whereas the steering camera is always



(a) **(b)**

Figure 6.7: *Comparison between the path computed (a) by steering a camera and (b) by using our rail positioning process.*

behind on the rail. This graph reveals the main drawback of using a reactive method like steering behaviors. The camera only moves or accelerates at the last possible moment for it does not anticipate the characters motion. Our solution on the other hand is able to anticipate the movements by globally optimizing the camera positions. Figure 6.9 illustrates the difference between the two approaches. When the characters suddenly move away from the camera, the reactive solution does not handle the abrupt movement and looses the visibility over the targets for a moment. Our camera is able to maintain the visibility over the characters by moving earlier along the rail.

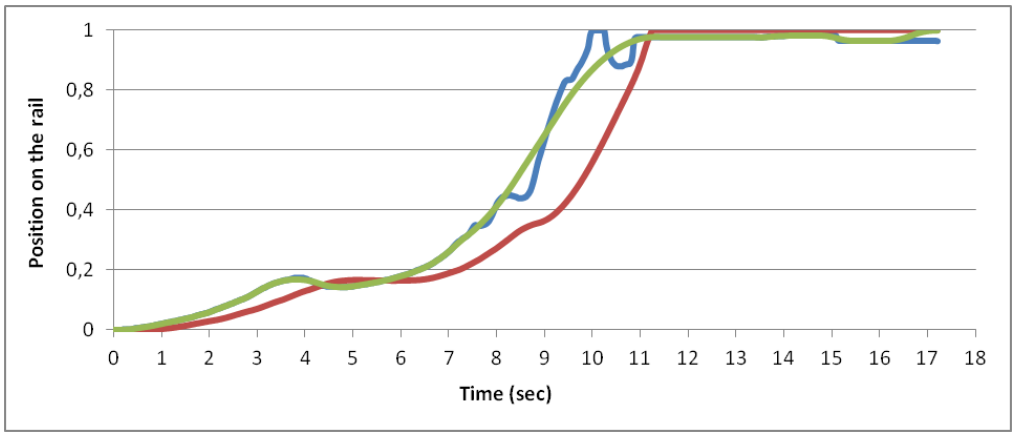


Figure 6.8: *Position of the cameras on the rail along time. Our optimized solution (green) smoothly approximates the optimal camera positions on the rail (blue) while the autonomous camera (red) hardly keeps-up and introduced a delay.*

6.5.3 Camera rails for cinematic replay

To further evaluate our new camera planning method, we tested it with the cinematic replay system detailed in chapter 5. To do so, we only changed part of the cinematographer component (see Figure 5.1). In the previous version, we used the PSL specification given by the camera behavior to prune regions of the manifold surface and steer the camera towards the remaining area on the manifold. Instead, we now create a new rail every time the PSL specification changes. The raw trajectory of each camera is computed by taking the middle point of the

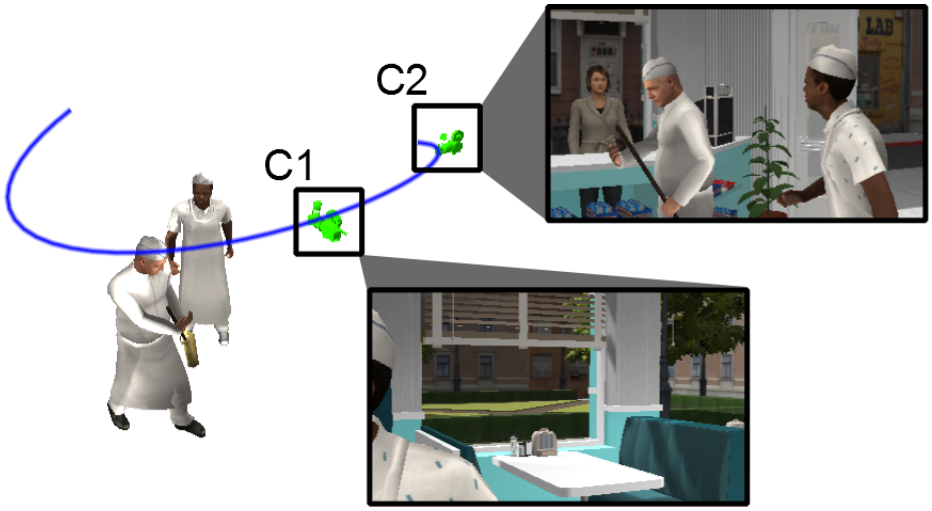


Figure 6.9: *Viewpoints of the two constrained cameras as the characters move. The autonomous camera $C1$ is not able to track them whereas our solution $C2$ is able to maintain the visibility over the two characters by anticipating their movements.*

remaining area on the manifold at each frame. The tests were also conducted with the game introduced in section 5.6 and with the same set of behaviors. Edited sequences are available online with the rest of the results[3].

(a) *First camera* **(b)** *Second camera* **(c)** *Master shot*

Figure 6.10: *Shots computed for three different camera behaviors on the same scene at the same time: (a) over-the-shoulder behavior defined on the main character Frank, (b) over-the-shoulder behavior defined on Olivia and (c) master shot behavior on Franck and Olivia.*

Figure 6.10 shows examples of shots recorded by the system during the replay. For dialogues sequences, the results are identical with our two methods. This is explained by the lack of motion of the characters when involved in such actions ; the camera does not need to move to maintain the framing. The main differences between the two methods occur while tracking the main character. When a player moves in the environment, he tends to produce many unnecessary movements (*i.e.* going back and forth, left and right or abruptly changing direction), leading to unnatural and complex trajectories. Using steering behaviors, our previous solution is bound to partially reproduce these trajectories, leading to undesirable and unrealistic motions. On the other hand, as illustrated in Figure 6.11, our virtual rails offer a smooth solution to this problem. Moreover, as seen in previous results and again illustrated in Figure 6.11, due to their reactive nature, steering based cameras are not always able to keep up with the charac-

[3] https://team.inria.fr/imagine/camera-on-rails/

ters which results in poor framing quality, whereas our optimization based approach manages to ensure the framing.

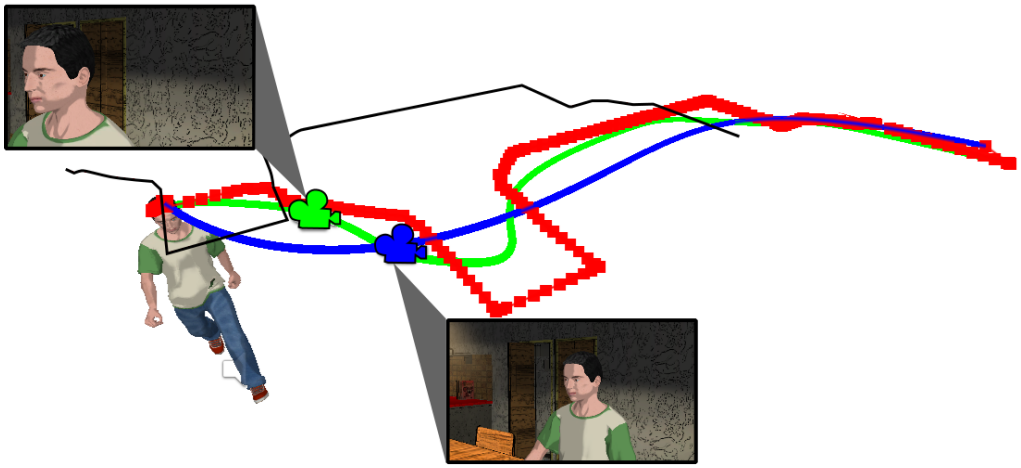


Figure 6.11: *The virtual camera rail (blue) provides a more realistic approximation of the raw trajectory (red) than the steering-based camera (green). The optimization process on the rail also results in better framing of the character.*

Applying our new method to the case of cinematic replays confirmed its efficiency and the quality of its results. – a user study is yet to be conducted to fully assess the benefits of the technique. However it also highlighted its main limitation. As we purposely limited camera rails to cubic Bezier curves, it is sometimes difficult to well approximate complex trajectories (see in Figure 6.12).

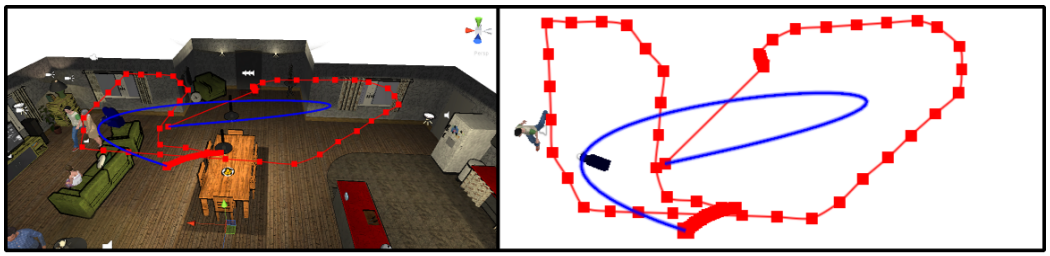


Figure 6.12: *The computed rail (blue), is not always capable to approximate the raw trajectory (in red) when characters' motions are too complex.*

6.5.4 Virtual movie making using virtual rails

To test our camera planning technique on a more complex situation, we used the dataset introduced in section 4.7.1. Unlike previous test cases, this scene not only provides complex characters' trajectories, but also stages a variety of animated actions. For instance, some dialogues between characters occur while they are moving, which makes the computation of some standard shots (such as Over-the-shoulder shots) much more complicated.

For this final experiment however, since there is no information available on the narrative importance of the characters, we could not use the cinematic replay system. Instead, we implemented a slightly different and more generic solution based on the concept of idioms. However,

unlike traditional idiom-based methods, the system does not execute the editing in realtime ; it only handles the cinematography and then uses our editing framework to perform the editing. As a result, this approach lightens the burden of creating idioms. This task no longer requires expert knowledge nor tremendous amount of work. To create an idiom, it is only necessary to specify the initial and final set-up (*i.e.* PSL specifications) of each camera ; these set-ups are used to compute the initial and final viewpoints of the rails.

To test our solution we devised four different idioms: monologue, two-person dialog, three-person dialog and walk. Each idiom is then instantiated with a starting time, an ending time, and the list of characters involved. For the scene from *Back to the Future* we instantiated the idioms of Figure 6.13 – we simply gathered related actions of the scenario.

```
Dialog2    (0     , 13.5  , Marty and George)
Dialog3    (13.5  , 37.75 , Marty and George and Goldie)
Monologue  (37.75 , 50.1  , Goldie)
Move       (46.0  , 56.25 , Goldie and Lou)
Dialog2    (56.25 , 62.5  , Goldie)
Move       (62.5  , 79.6  , Marty and George)
```

Figure 6.13: *List of idioms defined for an extract from* Back To The Future. *Each idiom is defined by its type, its starting time, its ending time and the list of characters involved.*

Figure 6.14 shows the scene with all the rails that were computed to shot the whole sequence. For each idiom two to three different rails are generated. In Figure 6.15, we show some of the viewpoints computed on the rails for different idioms. The complete edited sequence can be seen online, along with the other results.

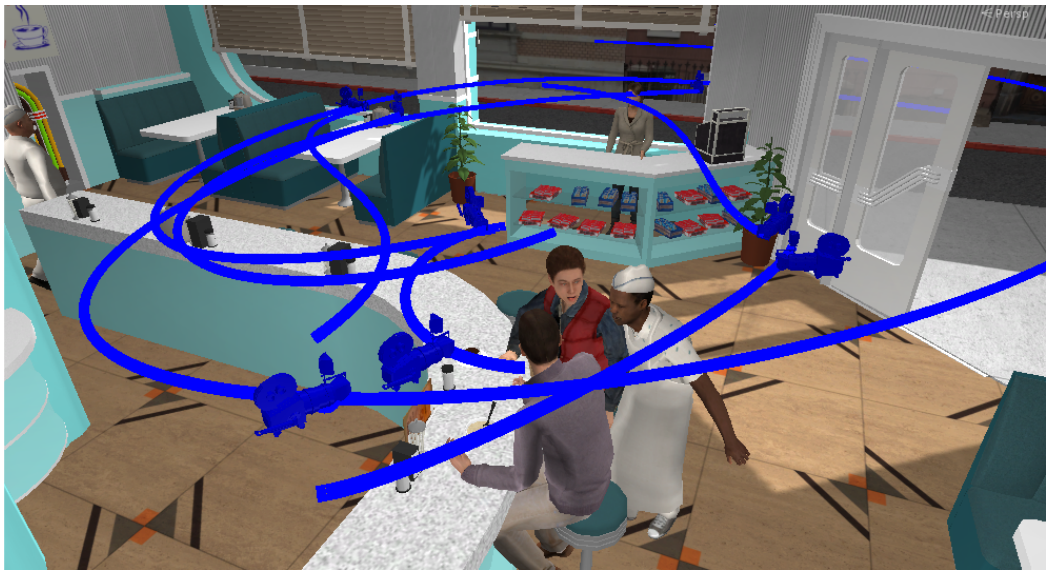


Figure 6.14: *Overview of the scene after the computation of all the rails. Two distinct rails are computed for each idiom.*

This last test however highlighted another limitation of our method. As seen in Figure 6.16, our model does not consider the environment and the potential obstacles that might get in the

(a) **(b)** **(c)** **(d)**

Figure 6.15: *Shot computed at different times for different idioms in* Back to the Future*: (a) two-person dialogue, (b) three-person dialogue, (c) monologue, (d) walk.*

way of the camera. Nevertheless, this problem is dealt with during the editing. Indeed, due to its poor visual quality, the resulting shot is not selected by the editing tool during the optimization process.

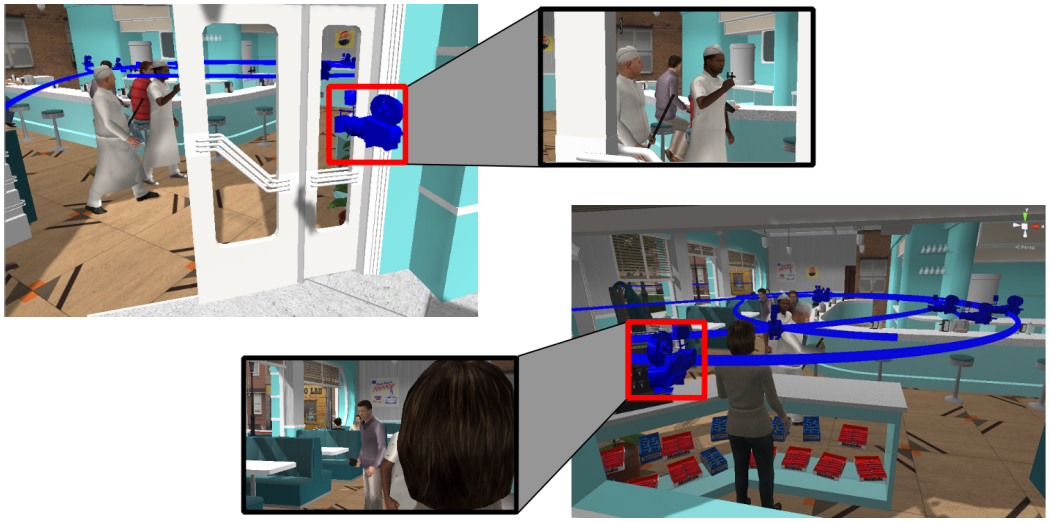


Figure 6.16: *In some cases, the rails are poorly placed in environment which results in unaesthetic shots. Objects or characters sometimes get in the way of the camera.*

6.6 Limitations and Future Work

Currently, our model clearly lacks the capacity to compute occlusion free paths. Future work could be conducted to account for environmental constraints during the computation of the rail from the raw trajectory, and during the optimization stage by using visibility tests to compute the optimal position on the rail. The work by Oskam *et al.* [OSTG09] presents interesting leads to address this issue. Another important limitation of the method resides in its objective to create simple camera rails. The use of cubic Bezier curves limits the complexity of the rail but also prevents it to approximate complex trajectories. To overcome this limitation, possible extensions of our method could be to combine pieces of camera rails together.

This work also offers new perspectives in terms of interactive tools. Allowing users to interact with camera rails (or other camera rigs) and refine camera placement along the rail represents a first step in the creation of tools to assist cinematographers and animators in rapidly drafting camera motions for applications such as virtual movie-making or cinematic replays.

6.7 SUMMARY

In this last chapter, we have introduced a novel approach to create smooth and natural camera motions that relies on traditional cinematographic techniques. Our method automatically computes a camera rail from the specification of initial and final framings, and by knowing the motions of the targets to track. We rely on two constrained-optimization processes that ensure both the proper framing of target characters and smooth changes in the camera speed. Results demonstrate the benefits of our technique in comparison with previous approaches. It also showed that combined with our automatic editing method (see chapter 4), the system automatically generates complete sequences from 3D animated content.

CHAPTER

7

CONCLUSION

Throughout this dissertation we have tackled the grand challenge that is the automatic computation of cinematographic sequences from 3D animated content. We did so by addressing both the cinematography and editing aspects of movie-making. We devised new algorithms and provided new tools to automatically compute camera shots and edit them together. In this final chapter we summarize all our contributions and propose some interesting perspectives for future work in the field.

7.1 CONTRIBUTIONS

Drawing inspiration from a technique commonly used in a different field of study – namely autonomous control of virtual agents – we proposed a new model for addressing the cinematography aspect of our grand challenge. Adapting Reynolds' model of steering behaviors to camera control in dynamic environments, we were able to devise new algorithms to coordinate both position and orientation of our camera agents in realtime. The separate use of forces and torques to handle respectively the position and the orientation allows a refined and physically plausible control of the camera. This physically-based approach demonstrated great results when tested in a crowd simulation.

After this first approach to cinematography problem, we addressed the lack of editing tools. Using continuity editing as a base for this work, we mathematically formalized its rules to quantify editing errors. Our solution minimizes such errors through an efficient dynamic-programming optimization. Based on a semi-Markov assumption, the model also allows us to overcome limitations of other offline approaches by providing control over the pacing of the sequence. The user study conducted in chapter 4 proved the capacity of our solution to properly edit camera shots together without making basic errors.

Once equipped with the cinematographic and editing tools, we tackled the challenge that is the generation of cinematic replay of game sessions. We devised a tool where the cinematography and editing are guided using narrative and geometric information extracted from the gameplay. Extending our first approach to camera control, we managed to improve the force-based method and formalize the definition of camera behaviors to satisfy given communicative goals. The system was tested on a dialogue-based serious game and showed promising results.

Finally, after highlighting the need for more realistic trajectories we introduced an offline method to constrain camera trajectories onto virtual camera rails. Unlike our previous approach this method is not suited for realtime cinematography. The motion planning is performed through a double optimization process on the position and orientation that ensures plausible motion – in terms of speed and acceleration – while globally maintaining the composition on the screen. Combined with the editing tool, this approach provided better results on the cinematic replay examples and also generated aesthetically pleasing sequences using the *Back To The Future* dataset.

Parts of these contributions have been published in peer reviewed conferences. Following is the list of publications:

1. GALVANE Q., CHRISTIE M., LINO C., RONFARD R.: Camera-on-rails: Automated Computation of Constrained Camera Paths. In *MIG 2015*. 8th ACM SIGGRAPH conference on Motion in Games.

2. GALVANE Q., RONFARD R., CHRISTIE M.: Comparing film-editing. In *WICED 2015*. Eurographics Workshop on Intelligent Cinematography and Editing.

3. GALVANE Q., RONFARD R., LINO C., CHRISTIE M.: Continuity Editing for 3D Animation. In *AAAI 2015*. AAAI Conference on Artificial Intelligence.

4. GALVANE Q., RONFARD R., CHRISTIE M., SZILAS N.: Narrative-Driven Camera Control for Cinematic Replay of Computer Games. In *MIG 2014*. 7th ACM SIGGRAPH conference on Motion in Games.

5. LINO C., RONFARD R., GALVANE Q., GLEICHER M.: How Do We Evaluate the Quality of Computational Editing Systems?. In *WICED 2014*. AAAI Workshop on Intelligent Cinematography And Editing.

6. GALVANE Q., CHRISTIE M., RONFARD R., LIM C.K., CANI M-P: Steering Behaviors for Autonomous Cameras. In *MIG 2013*. 6th ACM SIGGRAPH conference on Motion in Games.

7. LIM C.K., CANI M-P, GALVANE Q., PETTRÉ J., TALIB A.Z.: Simulation of Past Life: Controlling Agent Behaviors from the Interactions between Ethnic Groups. In Digital Heritage International Congress 2013.

7.2 PERSPECTIVES

With our final contribution, we presented a solution that automatically handles the shooting and editing of animated 3D environments. This last achievement represents a new step in the creation of tools dedicated to the production of virtual cinematographic content. To get virtual cinematography to the next level, many open challenges are yet to be addressed. We here propose several leads for future work based on our research that could greatly benefits this field of study.

Narrative Discourse

One of the main concerns of this thesis was to provide new means of virtual storytelling that better convey the narrative discourse. Both our work on camera control and editing used the narrative impact of actors and actions to guide their respective processes. However, there remains a lot of room for improvement on both sides to enrich our narrative models and cover a larger spectrum of communicative goals. The work conducted in chapter 5 provides a generic solution to narrative driven cinematography but doesn't make full use of the narrative information that might be available. For instance, future research on this field could explore richer model of information to account for emotions of the characters. The work by [KM02] was a first step in this direction. The comparison study conducted in chapter 4 also clearly highlighted the need for further investigation of richer narrative models to improve the expressiveness of editing tools. Research conducted in cognitive science, and more precisely in cognitive film theory [And96, Smi05], represents a valuable asset for future work. Indeed, the editing defines the cognitive path taken by the viewer – established by inferring information from transitions between consecutive shots – and thus its understanding of the story logic. Therefore, the design of more complex models requires a higher understanding of viewers' cognitive reasoning [JY09a, JY09b].

Live-action video editing

The editing model we detailed in this thesis is based on the minimization of a weighted sum of cost functions. The evaluation of these cost functions can be performed in screen space, and does not require knowledge of the three-dimensional configuration of actors in the scene. As a result, our model is equally applicable in principle to live-action video editing. Future work is however required to adapt the proposed algorithm to the case of live-action video, which will require the detection and recognition of actors in all rushes [GR13, Gan14] and the recognition of their actions [WRB11].

Learning from movies

The comparison analysis conducted in chapter 4 also stressed the conventional nature of our results and raised the question of learning variations of editing styles from real movies. As mentioned before, our model is based on the minimization of a weighted sum of cost functions. Thus, a given set of weights defines a variation of the continuity editing style. In order to extrapolate our model to other editing styles using existing movies, and in addition to the detection and recognition of actors, future work is needed for annotating movie scene examples with actions and their narrative importance. This tedious annotation task is essential for learning the statistical models of film styles from the training sets. It can be, at least partially, achieved through various automatic approaches either based on computer vision techniques [HFR06] or using movie scripts to extract the information [PGHA15].

User interactions

The research presented in this thesis specifically focused on the task of automating movie making processes with non-interactive approaches. By avoiding low-level interactions with users, we are able to generate multiple camera shots and perform basic edits without continuity errors. However, due to the complexity and the subjective nature of movie making – each director shoots and edits movies in different ways –, putting the user back in the loop seems a wise direction for future work. For instance, the solution introduced in the last chapter highlighted the potential of user interaction with virtual rails. Drafting an initial camera path that optimizes camera placement in space and time from minimal user input would indeed save animators a lot of time. Further investigation on interactive modification of trajectories accounting for optimal camera placement would also greatly ease the shot creation process. On the editing side, traditional approaches require editors to manually perform every cut of the movie. In chapter 4, we pointed out the capacity of our solution to consider user input to constrain camera choices. Improvements on our current system, especially with regards to user inputs and user interface, could greatly benefits editors. Combined with a more traditional interface, users could be allowed to handle important cuts and let our system fill in the rest.

Realtime Cinematography

A last potential direction is worth considering. In this thesis, we started to address camera control issues as a realtime challenge but then focused on offline algorithms to deal with the cinematography. The work detailed in chapter 3 and extended in chapter 5 probably needs more consideration as it could be adapted for realtime applications. Using online editing methods (such as [CAH*96]) to cut between cameras, our approach could be improved to provide the missing tools for fully automatic live cinematography. Finally, as seen in section 2.2.2, a large amount of work on realtime camera control has drawn its inspiration from research in robotics.

To this date however, the proposed reactive approaches clearly lack cinematographic quality. Going the other way around and use our work on realtime cinematography in the robotics field seems a promising perspective. In particular, with the recent interest in drones shown by both the robotics and cinema communities [Dia15, Aud14], our work on autonomous cameras represents an interesting lead for the control of autonomous drones.

BIBLIOGRAPHY

[ACOYL08] ASSA J., COHEN-OR D., YEH I.-C., LEE T.-Y.: Motion overview of human actions. In *ACM SIGGRAPH Asia* (2008), pp. 115:1–115:10.

[ADV02] ADAMS B., DORAI C., VENKATESH S.: Toward automatic extraction of expressive elements from motion pictures: tempo. *IEEE Transactions on Multimedia 4*, 4 (2002), 472–481.

[AKY05] AMERSON D., KIME S., YOUNG R. M.: Real-time cinematic camera control for interactive narratives. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology* (New York, NY, USA, 2005), ACE '05, ACM, pp. 369–369.

[And96] ANDERSON J.: *The Reality of Illusion: An Ecological Approach to Cognitive Film Theory*. Southern Illinois University Press,, 1996.

[AP12] ASCHER S., PINCUS E.: *The Filmmaker's Handbook: A Comprehensive Guide for the Digital Age*. Plume, 2012.

[Ari76] ARIJON D.: *Grammar of the Film Language*. Silman-James Press, 1976.

[Aud14] AUDRONIS T.: *Building Multicopter Video Drones*. Packt Publishing Ltd, 2014.

[AVF04] ANDÚJAR C., VÁZQUEZ P., FAIRÉN M.: Way-finder: Guided tours through complex walkthrough models. In *Computer Graphics Forum* (2004), vol. 23, pp. 499–508.

[AWCO10] ASSA J., WOLF L., COHEN-OR D.: The virtual director: a correlation-based online viewing of human motion. *Computer Graphics Forum 29*, 2 (2010), 595–604.

[BC11] BERLINER T., COHEN D. J.: The illusion of continuity: active perception and the classical editing system. *Journal of Film and Video 63*, 1 (2011), 44–63.

[BDER08] BURELLI P., DI GASPERO L., ERMETICI A., RANON R.: Virtual camera composition with particle swarm optimization. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2008), vol. 5166 LNCS, pp. 130–141.

[BDP99] BARRAL P., DORME G., PLEMENOS D.: Visual understanding of a scene by automatic movement of a camera. In *International Conference GraphiCon'99, Moscow (Russia), August 26 – September 3* (1999).

[BDP00] BARRAL P., DORME G., PLEMENOS D.: Scene understanding techniques using a virtual camera.

[BGBLT97] BECKER C., GONZÁLEZ-BAÑOS H., LATOMBE J., TOMASI C.: An intelligent observer. In *Experimental Robotics IV* (1997), Khatib O., Salisbury J., (Eds.), vol. 223 of *Lecture Notes in Control and Information Sciences*, Springer Berlin Heidelberg, pp. 151–160.

[BGL98] BARES W. H., GREGOIRE J. P., LESTER J. C.: Realtime constraint-based cinematography for complex interactive 3d worlds. In *In Tenth National Conference on Innovative Applications of Artificial Intelligence* (1998), pp. 1101–1106.

[BJ09] BURELLI P., JHALA A.: *Dynamic Artificial Potential Fields for Autonomous Camera Control*. AAAI Press, 2009.

[Bli88] BLINN J.: Where am i? what am i looking at? (cinematography). *Computer Graphics and Applications, IEEE 8*, 4 (July 1988), 76–81.

[BMBT00] BARES W. H., MCDERMOTT S., BOUDREAUX C., THAINIMIT S.: Virtual 3D camera composition from frame constraints. In *Proceedings of the eighth ACM international conference on Multimedia - MULTIMEDIA '00* (2000), pp. 177–186.

[Bor98] BORDWELL D.: *On the History of Film Style*. Harvard University Press, 1998.

[BT01] BORDWELL D., THOMPSON K.: *Film Art: An Introduction*. McGraw-Hill Higher Education, 2001.

[BZRL98] BARES W. H., ZETTLEMOYER L. S., RODRIGUEZ D. W., LESTER J. C.: Task-sensitive cinematography interfaces for interactive 3d learning environments. In *Proceedings of the 3rd International Conference on Intelligent User Interfaces* (New York, NY, USA, 1998), IUI '98, ACM, pp. 81–88.

[CAH*96] CHRISTIANSON D. B., ANDERSON S. E., HE L.-W., WELD D. S., COHEN M. F., SALESIN D. H.: Declarative camera control for automatic cinematography. In *AAAI* (1996), pp. 148–155.

[Can23] CANUDO R.: *Manifeste des sept arts*. Gazette des septs arts, 1923.

[CDN10] CUTTING J. E., DELONG J. E., NOTHELFER C. E.: Attention and the Evolution of Hollywood Film. *Psychological Science 21*, 3 (Mar. 2010), 432–439.

[CJBY08] CHEONG Y.-G., JHALA A., BAE B.-C., YOUNG R. M.: Automatically generating summary visualizations from game logs. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference* (2008), The AAAI Press.

[CJMT08] COUR T., JORDAN C., MILTSAKAKI E., TASKAR B.: Movie/script: Alignment and parsing of video and text transcription. In *Proceedings of 10th European Conference on Computer Vision, Marseille, France* (2008).

[CLR12] CHRISTIE M., LINO C., RONFARD R.: Film editing for third person games and machinima. In *Workshop on Intelligent Cinematography and Editing* (2012).

[CM01] COURTY N., MARCHAND E.: Computer animation: a new application for image-based visual servoing. In *IEEE Int. Conf. on Robotics and Automation, ICRA'01* (Seoul, South Korea, 2001), vol. 1, pp. 223–228.

[CN05] CHRISTIE M., NORMAND J.-M.: A Semantic Space Partitioning Approach to Virtual Camera Composition. *Computer Graphics Forum* (2005).

[CNO12] CHRISTIE M., NORMAND J.-M., OLIVIER P.: Occlusion-free camera control for multiple targets. In *Symposium on Computer Animation* (2012), pp. 59–64.

[CO09] CHRISTIE M., OLIVIER P.: Camera control in computer graphics: Models, techniques and applications. In *ACM SIGGRAPH ASIA 2009 Courses* (New York, NY, USA, 2009), SIGGRAPH ASIA '09, ACM, pp. 3:1–3:197.

[Con13] CONDAT L.: A Direct Algorithm for 1D Total Variation Denoising. *IEEE Signal Processing Letters 20*, 11 (2013), pp. 1054 – 1057.

[CPR98] CARRIVE J., PACHET F., RONFARD R.: Using Description Logics for Indexing Audiovisual Documents. In *International Workshop on Description Logics (DL '98)* (Trento, Italy, June 1998).

[CPR00] CARRIVE J., PACHET F., RONFARD R.: Clavis - a temporal reasoning system for classification of audiovisual sequences. In *Recherche d'Informations Assistée par Ordinateur (RIAO '00)* (Paris, France, Apr. 2000), Mariani J.-J., Harman D., (Eds.), pp. 1400–1415.

[DBC12] DELONG J. E., BRUNICK K. L., CUTTING J. E.: Film through the human visual system: Finding patterns and limits. In *Social Science of Cinema* (2012), Kaufman J. C., Simonton D. K., (Eds.), Oxford University Press.

[dDR98] D'YDEWALLE G., DESMET G., RENSBERGEN J. V.: Film perception: the processing of film cuts. In *Eye guidance anin reading and scene perception* (1998), Eslsevier Science Ltd.

[DeL09] DELOURA M.: *Real Time Cameras, A Guide for Game Designers and Developers*. Morgan Kaufman, 2009.

[Dia15] DIAZ T.: Lights, drone... action. *Spectrum, IEEE 52*, 7 (2015), 36–41.

[Dmy84] DMYTRYK E.: *On Film Editing: An Introduction to the Art of Film Construction*. Focal Press, 1984.

[Dru95] DRUCKER S. M.: *Intelligent Camera Control for Graphical Environments*. PhD thesis, Massachusetts Institute of Technology, 1995.

[DYR11a] DOMINGUEZ M., YOUNG R. M., ROLLER S.: Automatic identification and generation of highlight cinematics for 3d games. In *Proceedings of the 6th International Conference on Foundations of Digital Games* (2011), ACM, pp. 259–261.

[DYR11b] DOMINGUEZ M., YOUNG R. M., ROLLER S.: Design and evaluation of afterthought, a system that automatically creates highlight cinematics for 3d games. In *AIIDE* (2011), The AAAI Press.

[DZ94] DRUCKER S. M., ZELTZER D.: Intelligent camera control in a virtual environment. In *Proceedings of Graphics Interface '94* (1994), pp. 190–199.

[DZ95] DRUCKER S. M., ZELTZER D.: CamDroid: A system for implementing intelligent camera control. In *Proceedings of the Symposium on Interactive 3D Graphics* (1995), pp. 139–144.

[DZO*13] DAVIS N., ZOOK A., O'NEILL B., HEADRICK B., RIEDL M., GROSZ A., NITSCHE M.: Creativity support for novice digital filmmaking. In *SIGCHI Conference on Human Factors in Computing Systems* (2013), ACM, pp. 651–660.

[ECR92] ESPIAU B., CHAUMETTE F., RIVES P.: A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation 8* (1992).

[ER07] ELSON D. K., RIEDL M. O.: A lightweight intelligent virtual cinematography system for machinima generation. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE '07)* (2007).

[FF04] FRIEDMAN D., FELDMAN Y. A.: Knowledge-based cinematography and its applications. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004* (2004), pp. 256–262.

[Fis]

[Gan14] GANDHI V.: *Automatic Rush Generation with Application to Theatre Performances*. Theses, Grenoble University, Dec. 2014.

[Gd07] GERMEYS F., D'YDEWALLE G.: The psychology of film: perceiving beyond the cut. *Psychological Research 71*, 4 (2007), 458–466.

[Gen72] GENETTE G.: *Figures III*. Seuil, 1972.

[Ger09] GERAERTS R.: Camera planning in virtual environments using the corridor map method. In *Proceedings of the 2Nd International Workshop on Motion in Games* (Berlin, Heidelberg, 2009), MIG '09, Springer-Verlag, pp. 194–206.

[GKE11] GRUNDMANN M., KWATRA V., ESSA I.: Auto-directed video stabilization with robust l1 optimal camera paths. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011)* (2011).

[GR13] GANDHI V., RONFARD R.: Detecting and naming actors in movies using generative appearance models. In *CVPR* (2013).

[GW92] GLEICHER M., WITKIN A.: Through-the-lens camera control. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1992), SIGGRAPH '92, ACM, pp. 331–340.

[Haw05] HAWKINS B.: *Real-Time Cinematography for Games*. Charles River Media, 2005.

[HCS96] HE L.-W., COHEN M. F., SALESIN D. H.: The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *SIGGRAPH* (1996), ACM, pp. 217–224.

[HFR06] HILTON A., FUA P., RONFARD R.: Modeling people: Vision-based understanding of a person's shape, appearance, movement, and behaviour. *Comput. Vis. Image Underst. 104*, 2 (Nov. 2006), 87–89.

[HH09] HAIGH-HUTCHINSON M.: *Real Time Cameras: A Guide for Game Designers and Developers*. CRC Press, 2009.

[HHS01] HALPER N., HELBING R., STROTHOTTE T.: A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum 20*, 3 (2001), 174–183.

[HO00] HALPER N., OLIVIER P.: CamPlan: A Camera Planning Agent. In *Smart Graphics 2000 AAAI Spring Symposium* (2000), pp. 92–100.

[HSRD12] HABONNEAU N., SZILAS N., RICHLE U., DUMAS J.: 3D Simulated Interactive Drama for Teenagers coping with a Traumatic Brain Injury in a Parent. In *5th International Conference on International Digital Storytelling (ICIDS 2012). LNCS 7648* (Heidelberg, 2012), Oyarzun D., Peinado F., Young R. M., Elizalde A., Méndez G., (Eds.), Springer, pp. 174–182.

[Jha06] JHALA A.: Darshak: an intelligent cinematic camera planning system. In *AAAI'06: proceedings of the 21st national conference on Artificial intelligence* (2006), AAAI Press, AAAI Press, p. 1918–1919.

[JY09a] JHALA A., YOUNG R. M.: Comparing effects of different cinematic visualization strategies on viewer comprehension. In *Proceedings of the 2Nd Joint International Conference on Interactive Digital Storytelling: Interactive Storytelling* (Berlin, Heidelberg, 2009), ICIDS '09, Springer-Verlag, pp. 26–37.

[JY09b] JHALA A., YOUNG R. M.: Evaluation of intelligent camera control systems based on cognitive models of comprehension. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (New York, NY, USA, 2009), FDG '09, ACM, pp. 327–328.

[JY11] JHALA A., YOUNG R. M.: Intelligent machinima generation for visual storytelling. In *Artificial Intelligence for Computer Games*. Springer New York, 2011, pp. 151–170.

[JYM10] JHALA A., YOUNG R. M., MEMBER S.: Cinematic visual discourse: Representation, generation, and evaluation. *IEEE Transactions on Computational Intelligence and AI in Games* (2010).

[Kat09] KATZ S. D.: *Film Directing Shot by Shot: Visualizing from Concept to Screen*. Focal Press, 2009.

[KM02] KENNEDY K., MERCER R. E.: Planning animation cinematography and shot structure to communicate theme and mood. In *Smart Graphics* (2002), ACM, pp. 1–8.

[LC08] LI T.-Y., CHENG C.-C.: Real-time camera planning for navigation in virtual environments. In *Proceedings of the 9th international symposium on Smart Graphics* (Berlin, Heidelberg, 2008), SG '08, Springer-Verlag, pp. 118–129.

[LC12] LINO C., CHRISTIE M.: Efficient Composition for Virtual Camera Control. In *ACM Siggraph / Eurographics Symposium on Computer Animation* (Lausanne, Suisse, July 2012), Kry P., Lee J., (Eds.).

[LC15] LINO C., CHRISTIE M.: Intuitive and efficient camera control with the toric space. In *SIGGRAPH 2015* (2015), ACM Press.

[LCCR11] LINO C., CHOLLET M., CHRISTIE M., RONFARD R.: Computational Model of Film Editing for Interactive Storytelling. In *ICIDS 2011 - International Conference on Interactive Digital Storytelling* (Vancouver, Canada, Nov. 2011), Si M., Thue D., André E., Lester J. C., Tanenbaum J., Zammitto V., (Eds.), vol. 7069, Springer, pp. 305–308.

[LCG*13] LIM C. K., CANI M.-P., GALVANE Q., PETTRE J., TALIB A. Z.: Simulation of past life: Controlling agent behaviors from the interactions between ethnic groups. In *Digital Heritage International Congress* (2013), p. (to appear).

[LCL*10] LINO C., CHRISTIE M., LAMARCHE F., GUY S., OLIVIER P.: A Real-time Cinematography System for Interactive 3D Environments. In *SCA '10 Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Madrid, Spain, July 2010), pp. 139–148.

[LCRB11a] LINO C., CHRISTIE M., RANON R., BARES W.: The director's lens: An intelligent assistant for virtual cinematography. In *Proceedings of the 19th ACM International Conference on Multimedia* (New York, NY, USA, 2011), MM '11, ACM, pp. 323–332.

[LCRB11b] LINO C., CHRISTIE M., RANON R., BARES W.: A smart assistant for shooting virtual cinematography with motion-tracked cameras. In *ACM Mutlimedia (Technical Demonstration)* (2011), ACM Press.

[LGBBL97] LAVALLE S., GONZALEZ-BANOS H., BECKER C., LATOMBE J.-C.: Motion strategies for maintaining visibility of a moving target. *Proceedings of International Conference on Robotics and Automation 1* (1997).

[LLY*12] LEE T.-Y., LIN W.-C., YEH I.-C., HAN H.-J., LEE J., KIM M.: Social-event-driven camera control for multicharacter animations. *IEEE Transactions on Visualization and Computer Graphics 18*, 9 (2012), 1496–1510.

[LRGG14] LINO C., RONFARD R., GALVANE Q., GLEICHER M.: How Do We Evaluate the Quality of Computational Editing Systems? In *AAAI Workshop on Intelligent Cinematography And Editing* (2014).

[LSA01] LIMPERT E., STAHEL W. A., ABBT M.: Log-normal distributions across the sciences: Keys and clues. *BioScience 51*, 5 (2001), 341–352.

[LT00] LI T.-Y., TING H.-K. T. H.-K.: An intelligent user interface with motion planning for 3D navigation. *Proceedings IEEE Virtual Reality 2000 (Cat. No.00CB37048)* (2000).

[LZ14] LIXANDRU E. T., ZORDAN V.: Physical rig for first-person, look-at cameras in video games. In *Proceedings of the Seventh International Conference on Motion in Games* (New York, NY, USA, 2014), MIG '14, ACM, pp. 119–124.

[Mas65] MASCELLI J. V.: *The Five C's of Cinematography: Motion Picture Filming Techniques*. Silman-James Press, 1965.

[Mer10] MERCADO G.: *The Filmmaker's Eye: Learning (and Breaking) the Rules of Cinematic Composition*. Focal Press, 2010.

[MHJ95] MITCHELL C., HARPER M., JAMIESON L.: On the complexity of explicit duration hmm's. *IEEE Transactions on Speech and Audio Processing 3*, 3 (May 1995), 213–217.

[MJY09] MARTINEZ H. P., JHALA A., YANNAKAKIS G. N.: Analyzing the impact of camera viewpoint on player psychophysiology. In *Affective Computing and Intelligent Interaction and Workshops, 2009. ACII 2009. 3rd International Conference on* (2009), IEEE, pp. 1–6.

[MKSB11] MARKOWITZ D., KIDER J. T., SHOULSON A., BADLER N. I.: Intelligent camera control using behavior trees. In *Proceedings of the 4th International Conference on Motion in Games* (Berlin, Heidelberg, 2011), MIG'11, Springer-Verlag, pp. 156–167.

[MS02] MATEAS M., STERN A.: A behavior language for story-based believable agents. *IEEE Intelligent Systems 17*, 4 (July 2002), 39–47.

[Mue07] MUELLER E. T.: Modelling space and time in narratives about restaurants. *LLC 22*, 1 (2007), 67–84.

[Mue09] MUELLER E. T.: Automating commonsense reasoning using the event calculus. *Commun. ACM 52*, 1 (2009), 113–117.

[Mur86] MURCH W.: *In the blink of an eye*. Silman-James Press, 1986.

[Mur02] MURPHY K. P.: *Hidden semi-Markov models*. Tech. rep., MIT AI Lab, 2002.

[NO04] NIEUWENHUISEN D., OVERMARS M.: Motion planning for camera movements. *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 4* (2004).

[OGK*10] OZAKI M., GOBEAWAN L., KITAOKA S., HAMAZAKI H., KITAMURA Y., LINDEMAN R. W.: Camera movement for chasing a subject with unknown behavior based on real-time viewpoint goodness evaluation. *Vis. Comput. 26*, 6-8 (June 2010), 629–638.

[OHPL99] OLIVIER P., HALPER N., PICKERING J. H., LUNA P.: Visual Composition as Optimisation. In *Artificial Intelligence and Simulation of Behaviour* (1999), pp. 22–30.

[O'S09] O'STEEN B.: *The Invisible Cut: How Editors Make Movie Magic*. Michael Wiese Productions, 2009.

[OSTG09] OSKAM T., SUMNER R. W., THUEREY N., GROSS M.: Visibility transition planning for dynamic camera control. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, ACM, pp. 55–65.

[PGHA15] PAVEL A., GOLDMAN D., HARTMANN B., AGRAWALA M.: Sceneskim: Searching and browsing movies using synchronized captions, scripts and plot summaries. In *UIST* (2015).

[Pic02] PICKERING J. H.: *Intelligent Camera Planning for Computer Graphics*. PhD thesis, University of York, 2002.

[PKG12] PONTO K., KOHLMANN J., GLEICHER M.: Effective replays and summarization of virtual experiences. *IEEE Trans. Vis. Comput. Graph. 18*, 4 (2012), 607–616.

[PO03] PICKERING J. H., OLIVIER P.: Declarative camera planning roles and requirements. In *Proceedings of the 3rd International Conference on Smart Graphics* (Berlin, Heidelberg, 2003), SG'03, Springer-Verlag, pp. 182–191.

[RCU10] RANON R., CHRISTIE M., URLI T.: Accurately measuring the satisfaction of visual properties in virtual camera control. In *in Proceedings of the 2010 Smartgraphics Symposium* (2010), LNCS, Smartgraphics, Springer.

[Red15] REDFERN N.: The log-normal distribution is not an appropriate parametric model for shot length distributions of hollywood films. *Literary and Linguistic Computing 30*, 1 (2015), 137–151.

[Rey99] REYNOLDS C.: Steering behaviors for autonomous characters. In *Game Developers Conference* (1999), pp. 763–782.

[RGB13] RONFARD R., GANDHI V., BOIRON L.: The Prose Storyboard Language: A Tool for Annotating and Directing Movies. In *2nd Workshop on Intelligent Cinematography and Editing part of Foundations of Digital Games - FDG 2013* (Chania, Crete, Grèce, May 2013).

[ROF92] RUDIN L. I., OSHER S., FATEMI E.: Nonlinear total variation based noise removal algorithms. *Phys. D 60*, 1-4 (Nov. 1992), 259–268.

[Ron04] RONFARD R.: Reading movies: An integrated dvd player for browsing movies and their scripts. In *Proceedings of the 12th Annual ACM International Conference on Multimedia* (New York, NY, USA, 2004), MULTIMEDIA '04, ACM, pp. 740–741.

[Ron10] RONFARD R.: *Automated cinematographic editing tool*. Canadian Patent Application, Xtranormal Technologies, 2010.

[Ron12] RONFARD R.: A Review of Film Editing Techniques for Digital Games. In *Workshop on Intelligent Cinematography and Editing* (Raleigh, United States, May 2012), Arnav Jhala R. M. Y., (Ed.), ACM.

[RRTS15] ROHRBACH A., ROHRBACH M., TANDON N., SCHIELE B.: A dataset for movie description. *CoRR abs/1501.02530* (2015).

[RS14] RONFARD R., SZILAS N.: Where story and media meet: computer generation of narrative discourse. In *Computational Models of Narrative* (Quebec City, Canada, July 2014), Schloss Dagstuhl OpenAccess Series in Informatics.

[RT03] RONFARD R., THUONG T.: A framework for aligning and indexing movies with their script. In *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on* (July 2003), vol. 1, pp. I–21–4 vol.1.

[RU14] RANON R., URLI T.: Improving the efficiency of viewpoint composition. *IEEE Trans. Vis. Comput. Graph. 20*, 5 (2014), 795–807.

[SAB12] SERIN E., ADALI S. H., BALCISOY S.: Automatic path generation for terrain navigation. *Computers & Graphics 36*, 8 (2012), 1013–1024.

[Sal09] SALT B.: *Film Style and Technology: History and Analysis (3 ed.)*. Starword, 2009.

[Sal11] SALT B.: The metrics in cinemetrics. http://www.cinemetrics.lv/metrics_in_cinemetrics.php, 2011.

[SC04] SARAWAGI S., COHEN W. W.: Semi-markov conditional random fields for information extraction. In *Advances in Neural Information Processing Systems* (2004).

[SGLM03] SALOMON B., GARBER M., LIN M. C., MANOCHA D.: Interactive navigation in complex environments using path planning. In *Proceedings of the 2003 symposium on Interactive 3D graphics - SI3D '03* (2003), p. 41.

[Sha82] SHARFF S.: *The elements of cinema. Towards a theory of cinesthetic impact.* Columbia University Press, 1982.

[Smi05] SMITH T. J.: *An Attentional Theory of Continuity Editing*. PhD thesis, University of Edinburgh, 2005.

[SP08] SOKOLOV D., PLEMENOS D.: Virtual world explorations by using topological and semantic knowledge. *The Visual Computer 24*, 3 (2008), 173–185.

[SWCS08] SHI Q., WANG L., CHENG L., SMOLA A.: Discriminative human action segmentation and recognition using. semi-markov model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2008).

[TB93] THOMPSON R., BOWEN C. J.: *Grammar of the Edit*. Focal Press, 1993.

[TB98] THOMPSON R., BOWEN C. J.: *Grammar of the Shot*. Focal Press, 1998.

[TBN00] TOMLINSON B., BLUMBERG B., NAIN D.: Expressive autonomous cinematography for interactive virtual environments. In *International Conference on Autonomous Agents* (2000), AGENTS '00, ACM, pp. 317–324.

[TS67] TRUFFAUT F., SCOTT H. G.: *Truffaut/Hitchcock*. Simon & Schuster, 1967.

[VMGL12] VO C., MCKAY S., GARG N., LIEN J.-M.: Following a group of targets in large environments. In *Motion in Games* (2012), pp. 19–30.

[VS03] VÁZQUEZ P. P., SBERT M.: Automatic indoor scene exploration. In *Proceedings of 6th International Conference on Computer Graphics and Artificial Intelligence 3IA'2003* (Limoges (France), May 2003), pp. 13–24.

[WO90] WARE C., OSBORNE S.: Exploration and virtual camera control in virtual three dimensional environments. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics* (New York, NY, USA, 1990), I3D '90, ACM, pp. 175–183.

[WRB11] WEINLAND D., RONFARD R., BOYER E.: A survey of vision-based methods for action representation, segmentation and recognition. *Comput. Vis. Image Underst. 115*, 2 (Feb. 2011), 224–241.

[YMJ10] YANNAKAKIS G. N., MARTÍNEZ H. P., JHALA A.: Towards affective camera control in games. *User Modeling and User-Adapted Interaction 20*, 4 (Oct. 2010), 313–340.

[Yu10] YU S.-Z.: Hidden semi-markov models. *Artificial Intelligence 174*, 2 (2010), 215 – 243.

[ZKSC85] ZEMECKIS R., KERAMIDAS H., SCHMIDT A. F. E., CUNDRY D. C.: *Back to the future*. Universal Pictures, 1985.

[ZTM*07] ZEN H., TOKUDA K., MASUKO T., KOBAYASIH T., KITAMURA T.: A hidden semi-markov model-based speech synthesis system. *IEICE - Trans. Inf. Syst. E90-D*, 5 (May 2007), 825–834.