Real-time virtual cinematography for target tracking

Ludovic Burg

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Ludovic BURG

Real-time Virtual Cinematography for Target Tracking

Thèse présentée et soutenue à Rennes, le 10/11/2022
Unité de recherche : IRISA UMR CNRS 6074

Rapporteurs avant soutenance :

Tsai-Yen Li, Professeur, National Chengchi University
Rémi Cozot, Professeur, Université du Littoral

Composition du Jury :

Président :	Eric Marchand, Professeur, University Rennes, Inria, CNRS, IRISA
Examinateurs :	Tsai-Yen Li, Professeur, National Chengchi University
	Rémi Cozot, Professeur, Université du Littoral
	Eric Marchand, Professeur, University Rennes, Inria, CNRS, IRISA
	Hui-Yin Wu, Maître de Conférences, Université Côte d'Azur, Inria
	Roberto Ranon, Maître de Conférences, University of Udine
	Marc Christie, Maître de Conférences, University Rennes 1
Dir. de thèse :	Franck Multon, Professeur, University Rennes 2, Inria

ACKNOWLEDGEMENT

Before starting this section, I want to say that I am not good with this kind of thing and I want to apologise if I forget anyone.

First, I would like to thank my advisors, Marc Christie, Christophe Lino and Franck Multon for all the opportunities they gave me during the 3 years of this PhD. Even with the COVID thing, we could travel and collaborate with a many people and all were valuable experiences and memories that will stay with me all my life.

Then, I want to thank the reviewers for all the interesting comments and suggestions they gave me, and the jury members, for their questions and discussions during and after the defense.

I also want to thank all the friends I met during and before the PhD. Pierre, Florian, Maxime and all the J7 members. The friends from the Mimetic and Hybrid teams for the one beers, the outings, and all the events they have organised. Especially, Anthony, Alberto, Gwendal, Arzelenne as we started as intern around the same time, Niels for the badminton, Rebecca, Benjamin, Diane, Amaury. Also, the friends we share the office with, Xi and Xiaoyuan.

Finally, I want to thank everyone that we share drinks and quality time with.

RÉSUMÉ EN FRANÇAIS

L'histoire de l'art du cinéma remonte à l'année 1888 avec la création de la première caméra (Kinematograph) par Thomas Edison et du premier Cinématographe en 1895 par Auguste et Louis Lumière. Depuis le premier film enregistré intitulé ***Roundhay Garden Scene*** par *Louis Le Prince* en 1888, de nombreux films ont été réalisés. Tout au long de l'histoire du cinéma, le processus de création d'un film a été perfectionné de manière itérative par les cinéastes. Dans ce processus créatif et exploratoire, la chaîne de production a été divisée en plusieurs étapes [Ste05] :

— **L'écriture** : l'étape où le scénario du film est écrit.

— **Le développement** : l'étape où l'on décide si le film sera produit.

— **La préproduction** : l'étape où tous les préparatifs sont menés avant le tournage proprement dit, *e.g.* , storyboarding, construction du plateau de tournage, prévisualisation.

— **La production** : l'étape où les scènes sont filmées, et les éléments sonores sont enregistrés ou créés.

— **La postproduction** : l'étape qui regroupe les opérations qui finalisent la production : montage, création VFX, mixage sonore, étalonnage.

Tout au long du processus de réalisation d'un film, à l'instar des romanciers qui écrivent des livres en utilisant des mots, les cinéastes et leur équipe disposent de leur propre langage et de leurs propres moyens d'expression pour transmettre des émotions, une ambiance et des informations à travers l'écran [Eva20]. Nous énumérons ici certains des principaux procédés narratifs utilisés par les cinéastes :

— **L'éclairage** définit l'aspect et l'atmosphère du film, et les configurations et propriétés de l'éclairage peuvent avoir un impact sur l'image de nombreuses façons : *e.g.* le nombre de lumières, leur taille, leur forme, la distance et l'angle avec la scène.

— **Composition du plan** définit les positions et les relations à l'écran entre les éléments composant les plans.

— **Mouvement de la caméra** définit à quelle classe de mouvements appartient le mouvement de la caméra, beaucoup d'entre elles ont été définies au fil des années (*e.g.* travelling, tenu à la main, etc...). Ces mouvements sont souvent utilisés pour susciter une émotion ou mettre visuellement l'accent sur une partie de l'histoire (*cf.* [TB09b]).

— **Montage** définit les transitions entre les différents plans pris par les différentes caméras. Le montage définit le rythme du film, et aide le spectateur à se construire une représentation temporelle et spatiale cohérente de la scène.

— **Le son** définit une atmosphère et est crucial pour l'immersion du spectateur.

Tout au long de la longue histoire du cinéma, les ordinateurs ont joué un rôle essentiel dans l'amélioration de la qualité de la production. De nouveaux outils ont été conçus pour accélérer et faciliter toutes les étapes de la production, influençant non seulement le contenu du film, mais aussi la manière dont les films sont créés.

C'est notamment le cas du domaine de l'informatique graphique (CG) qui permet la création d'environnements virtuels réalistes ou non, dans lesquels les artistes disposent de nouveaux moyens d'expression. La forme des éléments 3D est créée sous forme de **modèle** 3D, puis ils sont **animés** pour définir leur déplacement et l'évolution de leur forme dans l'environnement 3D, et enfin ils sont affichés avec des techniques de **rendu** pour définir leur apparence.

Il est intéressant de noter que le domaine de l'image de synthèse a ouvert les portes à de nouveaux médias narratifs qui peuvent s'inspirer des éléments de la cinématographie (caméra, lumière, etc.) pour créer des expériences audiovisuelles différentes, telles que des jeux vidéo, des films d'animation par ordinateur, des expériences XR interactives. La transposition des connaissances cinématographiques à un environnement virtuel est appelée **Cinématographie virtuelle** depuis 1999, date à laquelle le terme est apparu [Fee04].

Applications de cinématographie virtuelle

En effet, les améliorations des technologies matérielles et logicielles ont eu un fort impact sur l'industrie du divertissement, en particulier sur les médias narratifs comme le cinéma et les jeux vidéo. Ces améliorations ont permis d'ajouter des éléments d'images de synthèse aux films, de réaliser des films entièrement en images de synthèse (films

d'animation par ordinateur) ou des productions narratives interactives (jeux vidéo, expériences de réalité augmentée ou de réalité virtuelle).

Dans le domaine du cinéma, des outils ont été créés pour faciliter le processus de **préproduction**. C'est le cas des outils de **prévisualisation** (Figure 1.1a). La conception d'un film se fait sur la base d'une scène de synthèse rendue à bas prix afin de planifier la composition, le cadrage, la position de la caméra et les mouvements des plans. Ce procédé permet de réduire le temps passé sur le plateau et donc de réduire les coûts de production.

De plus, aujourd'hui, les techniques de prévisualisation sont également utilisées sur les sets de tournages pour permettre aux équipes de placer les accessoires, les rails de caméra et les lumières en fonction des directives de préproduction, en utilisant des technologies de réalité virtuelle ou augmentée (par exemple l'outil *Cyclops* de *The Third Floor*, *cf.* Figure 1.1b) pour montrer où les éléments de synthèse seront ajoutés sur le plateau, par exemple des monstres, des explosions.

Dans les applications narratives, la caméra joue un rôle important dans l'expérience de l'utilisateur ou du spectateur. Dans les applications interactives telles que les jeux vidéo, les utilisateurs contrôlent généralement la caméra et les éléments du jeu, ce qui peut être une source de frustration. Pour un utilisateur novice, contrôler tous les éléments du jeu, simultanément avec la caméra, peut en effet s'avérer une tâche difficile. Même pour les utilisateurs expérimentés, dans les jeux hautement dynamiques, l'utilisateur peut coincer la caméra dans l'environnement. L'une des solutions pour éviter cela est d'automatiser le placement de la caméra. Lorsqu'elle est correctement automatisée, la caméra peut aider à conduire la narration elle-même, comme dans les films. Pourtant, l'automatisation de cette caméra d'une manière qui maintient l'expérience de l'utilisateur et la jouabilité reste un problème ouvert. La plupart des productions conçoivent à l'avance les placements et les mouvements de la caméra et, au moment de l'exécution, décident de ceux qui seront utilisés.

En guise de référence, voici quelques exemples de jeux qui ont fait des efforts pour inclure des éléments cinématographiques comme une grande partie de la narration : *Vagrant Story* (2000), développé par *Square Product*, qui est l'une des premières grandes productions à utiliser des caméras automatisées et à faire un gros effort sur les angles de caméra et le montage, tant dans les cinématiques que durant le jeu. Plus récemment, des jeux comme *The Last Of Us* (2013) par *Naughty Dog*, ou *God Of War* (2018) par *Santa Monica Studio*, ont un modèle de contrôle de caméra hybride qui est capable de

(a) Exemple de prévisualisation du court-métrage *mort en ré mineur* ; à gauche : les résultats de la prévisualisation ; à droite : le film

(b) Exemple d'outil de prévisualisation sur le set de tournage : *Cyclops* de *The Third Floor*

faire respecter le point de vue dans le jeu à des fins de narration entre les séquences de jeu avec un contrôle de caméra libre par l'utilisateur qui, malgré ces succès, nécessitent une quantité importante de travail manuel, un certain nombre de questions doivent être abordées.

En effet, une caméra et l'image qu'elle produit représentent une fenêtre sur le monde et son but est de construire une représentation informative, cohérente et esthétique dans l'esprit du spectateur, mais aussi de construire des représentations hypothétiques et émotionnelles plus complexes à court et long terme. Si cet objectif est bien atteint dans la cinématographie réelle grâce à un savoir-faire orchestré, il ne l'est pas encore dans les approches computationnelles. De plus, nous ne sommes pas en mesure aujourd'hui de créer des caméras émotionnelles et narratives de manière automatisée. La communauté a progressé vers cet objectif, du moins partiellement, et nous présentons dans le chapitre

2 un rapport sur les progrès récents dans le domaine, et identifions les défis de demain en gardant ces perspectives à l'esprit.

La conception de systèmes de caméra interactifs et automatisés couvre en effet un large éventail de défis. Nous proposons le terme de **cinématographie virtuel calculatoire** pour englober toutes les techniques numériques liées à la cinématographie interactive ou automatisée dans des environnements synthétiques qui reposent sur des approches informatiques.

Suivies esthétiques de cible : définition du problème

Dans cette thèse, nous proposons d'étudier un sous-problème spécifique de la cinématographie virtuelle computationnelle : le **suivi esthétique de cibles**. Dans les jeux vidéo, la tâche de suivi de caméra consiste à maintenir le personnage principal à l'écran tout en assurant d'autres propriétés dépendant du gameplay. Par exemple, dans les jeux d'action à la troisième personne, la caméra parvient à se concentrer sur des objectifs tels que des ennemis ou des éléments interactifs tout en maintenant le personnage principal à l'écran.

Dans les tournois de jeux compétitifs multijoueurs (*i.e.* e-sport) ou durant les matches de sport qui sont diffusés en ligne ou commentés lors d'événements en direct, les replays sont souvent utilisés pour aider les spectateurs à comprendre des actions spécifiques. Ici, la tâche de suivi doit transmettre les événements intéressants de manière compréhensible et esthétique.

De nombreux défis se posent lorsqu'on s'attaque à ce problème.

Défi 1 : visibilité et collision Le premier défi du suivi est de maintenir la visibilité à l'écran de la cible tout en évitant les collisions avec l'environnement. Il existe un certain nombre de solutions pour mesurer la visibilité (*e.g.* ray-casting, rasterization, visibility maps, ...) mais il est toujours difficile de mesurer la visibilité **de manière efficace** dans des environnements dynamiques. De plus, il est difficile de réagir à une occlusion : il faut mesurer la visibilité du voisinage local, et dans certains cas des régions plus larges autour de la cible. Le choix de l'endroit où la caméra se déplace pour éviter une occlusion est crucial car, dans certains cas, la meilleure solution est d'être occulté pendant une courte période pour atteindre une zone sans occlusion plutôt que de rester dans une zone qui sera occultée pendant une plus longue période.

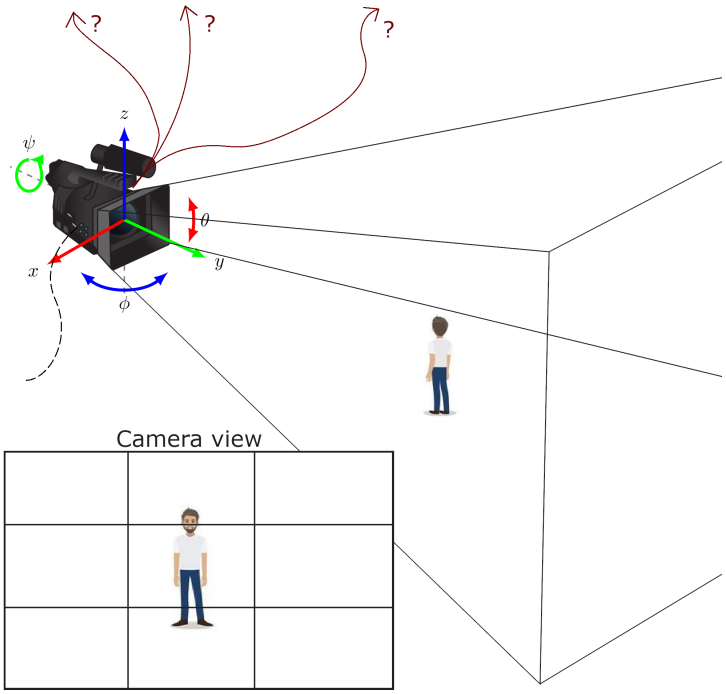


FIGURE 2 – Problème de contrôle de la caméra pour le suivi esthétique de la cible : utiliser les 6 degrés de liberté de la caméra pour maintenir la visibilité sur la cible, en assurant une composition de l'image et un mouvement esthétique de la caméra.

Défi 2 : dimension esthétique L'esthétique d'une trajectoire de caméra est portée par de nombreuses propriétés (angle de vue, distance de vue, composition de l'image) qui ne sont pas toujours compatibles (pour éviter une occlusion, il sera difficile de conserver un angle de vue ou une distance de vue). De plus, à un niveau d'expression supérieur, la notion d'esthétique est difficile à formaliser en termes de métriques à optimiser, car elle est subjective et dépendante du contexte offert par le contenu.

Défi 3 : dimension dynamique La nature dynamique de l'environnement traversé par la caméra ajoute une dimension de non-déterminisme dans les mouvements de la cible et des occultant. Cet aspect du problème rend difficile la planification à long terme du mouvement de la caméra, de l'occlusion de la cible et d'un mouvement esthétique de la caméra.

Défi 4 : Coordination en temps réel Enfin, ce qui rend la tâche globale de suivi esthétique des cibles pour les applications interactives particulièrement difficile, c'est que chaque aspect mentionné ci-dessus (visibilité, esthétique, dynamique) doit être évalué et équilibré en temps réel.

La question que nous abordons dans cette thèse est donc : "***Comment pouvons-nous contrôler efficacement et automatiquement une caméra virtuelle pour suivre une ou plusieurs cibles dans un environnement dynamique 3D ?***".

Nous présentons deux travaux portant sur la manière de contrôler automatiquement la caméra (Figure 1.2) afin de faire respecter des propriétés (*i.e.* collision, occlusion, distance de vue, angle de vue, saccade de la trajectoire de la caméra, cadrage, etc...) qui sont les vecteurs des règles cinématographiques pour la cinématographie virtuelle.

Publications

Au cours de cette thèse, nous avons publié deux articles de conférence, énumérés ci-dessous :

Real-time Anticipation of Occlusions for Automated Camera Control in Toric Space (Conference Paper) Ludovic Burg, Christophe Lino, Marc Christie. In Computer Graphics Forum 39, 2020 [BLC20].

Real-Time Cinematic Tracking of Targets in Dynamic Environments (Conference Paper) Ludovic Burg, Christophe Lino, Marc Christie. In GI 2021-Graphics Interface conference, 2021 [BLC21].

Grandes lignes

Dans le chapitre 2, nous fournissons une vue approfondie du domaine de recherche de la cinématographie virtuelle dans lequel nous montrons les avancées récentes dans les domaines.

En s'appuyant sur notre analyse du domaine de la cinématographie virtuelle, nous présentons une première contribution dans le chapitre **??**. Nous proposons de nous appuyer sur une approche de planification locale (**??**) pour résoudre le problème de l'évitement automatique des collisions et des occlusions par la caméra. Le système s'appuie sur l'espace torique présenté dans le chapitre 2 et sur la technique de rendu pour atteindre des performances en temps réel. Malgré ses avantages, le système présente des limitations liées à la nature locale de l'approche (la caméra peut rester bloquée dans des minima locaux).

Pour surmonter ces limitations, nous proposons une deuxième approche dans le chapitre 4. Sur la même base, nous proposons un système de caméra automatique qui s'appuie sur une approche hybride (*cf.* section 2.4.3). Dans cette seconde approche, au lieu d'utiliser un espace paramétrique existant (espace torique), nous introduisons un nouvel espace, le **camera animation space**. Nous utilisons le ray tracing GPU pour évaluer efficacement les mouvements dans l'espace d'animation avec une métrique de qualité de trajectoire personnalisée.

Dans le chapitre 5, nous conclurons cette thèse en résumant les contributions et leurs limites et parlerons des perspectives de cette recherche.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The story of the Art of **Motion picture** or cinematography goes back to the year of 1888 with the creation of the first camera (Kinematograph) by Thomas Edison and the first Cinematograph in 1895 by Auguste and Louis Lumière. From the first recorded movie titled ***Roundhay Garden Scene*** by *Louis Le Prince* in 1888, numerous movies were made. Across the movie history, the movie making process has been honed iteratively by moviemakers. In this creative and exploratory process, the production pipeline has been divided into multiple steps [Ste05]:

— **The writing**: the step where the scenario of the film is written.

— **The pre-production**: the step where all the preparations are conducted before the actual shooting, *e.g.* , storyboarding, building of the shooting set, previsualization.

— **The production**: the step where the scenes are filmed, and the sound elements are recorded or created.

— **The post-production**: the step that groups the operations that finalize the production: editing, VFX creation, sound mixing, color grading.

During the whole movie making process, similarly to novelists who write books using words, moviemakers and their team have their own language and means of expression to convey emotions, mood and information through the screen [Eva20] We list here some key narrative devices used by moviemakers:

— **Lighting** defines the look and feel of the movie, and there are many ways in which the lighting configurations and properties can impact an image: *e.g.* the number of lights, their size, shape, distance and angle with the scene, softness or indirect nature.

— **Shot composition** defines the positions and on-screen relations between the elements composing the shots.

— **Camera movement** defines to which class of motions the camera's movement belong and many of them have been defined across the years (*e.g.* travelling,

hand held, etc...). Such movements are often used to elicit emotion or bring focus to a part of the visual story (*cf.* [TB09b]).

— **Editing** defines the transitions between the different shots taken by the cameras. Editing defines the rhythm of the movie, as well as helps the viewer to build a coherent temporal and spatial representation of the scene.

— **The sound** defines an atmosphere and is crucial for the viewer's immersion.

Throughout the long history of cinema, computers played an essential role in improving the quality of the production. New tools were designed to help all production stages get faster and easier, not only influencing the contents that are in the film, but also influencing the way films are created.

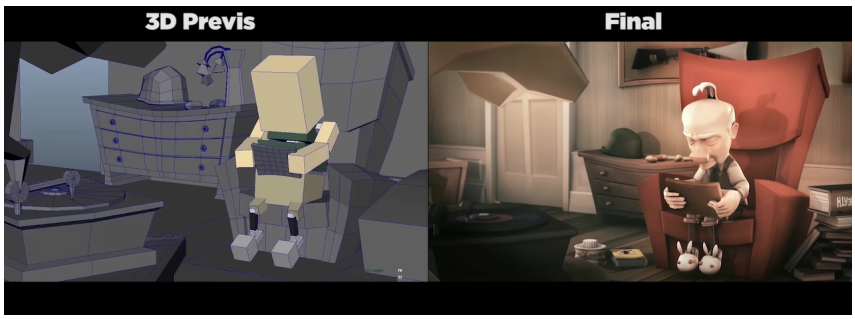
It is especially the case for the field of Computer Graphics (CG) that enables the creation of realistic or non-realistic virtual environments, in which artists get new means of expressions. The shape of 3D elements is created as a 3D **model**, then they are **animated** to define their displacement and shape evolution in the 3D environment, and finally they are displayed with **rendering** techniques to define their appearance.

Interestingly, the field of CG opened doors for new storytelling media which can draw from elements of cinematography (*e.g.* camera, light, etc...) to create different audiovisual experiences, such as video games, computer-animated films, interactive XR experiences. Transposing cinema knowledge to virtual environment has been called **Virtual cinematography** since 1999 when the term emerged [Fee04].

1.1 Virtual cinematography applications

Indeed, The improvements in hardware and software technologies have strongly impacted the entertainment industry, particularly on the storytelling media like cinema and video games. These improvements allowed adding CG elements to films, make fully CG films (*i.e.* computer animated films) or interactive storytelling productions (*e.g.* video games, AR or VR experiences).

In films, tools have been made to help the **pre-production** process. That is the case for the **previsualization** tools (Figure 1.1a). A movie is designed on a cheap rendered CG scene to plan the composition, framing, camera position and movement for the shots. This process allows to reduce the time spent on set and thus reduce production costs.



(a) Previsualization example from the short film *death in D minor*; Left: the previsualization results; right: the actual film

(b) On set previsualization tool example: *Cyclops* from *The Third Floor*

Furthermore, today the previsualization techniques are also used on set for the team to place props, camera rails or lights according to the post-production guidelines, using virtual or augmented reality technologies (*e.g.* the *Cyclops* tool from *The Third Floor*, *cf.* Figure 1.1b) to show where CG elements will be added on set, *e.g.* monsters, explosions.

For storytelling applications, the camera plays an important role in the user or viewer experience. In interactive applications such as video games, the users usually control the camera and the game elements, which can be a source of frustration. For a novice user controlling all game elements, simultaneously with the camera, can indeed be a challenging task. Even experimented users, in highly dynamic games the user can get the camera stuck in the environment. One of the solutions to avoid this is to automate the camera placement. When properly automated, the camera can help to drive the sto-

rytelling itself similarly to films [TB09b]. Yet, automating this camera in a way that maintains the user experience and the playability remains an open problem. Most productions actually pre-design camera placements and motions, and at runtime decide which ones to use.

As reference, some examples of games that made efforts to include cinematographic elements as a big part of the storytelling are: *Vagrant Story* (2000) developed by *Square Product*, which is one of the first big production with automated cameras and with a great effort put on camera angles and editing, both in cut scenes and in-game. More recently, games like *The Last Of Us* (2013) by *Naughty Dog*, or *God Of War* (2018) by *Santa Monica Studio*, have a hybrid camera control model that is able to enforce in-game point of view for storytelling purpose between gameplay sequences with a free camera control by the user which, despite these successes, require a significant amount of manual work, a number of issues need to be addressed.

Indeed, a camera and the image it yields, represent a window on the world and its purpose is to build an informative, coherent and aesthetic representation in the spectators mind, but also build more complex short and long term hypothetical and emotional representations. While this is well achieved in real cinematography through orchestrated craftsmanship, this is not reached yet in computational approaches. Moreover, we are not able to create emotional and narrative cameras in automated ways today. The community has been progressing towards this goal at least partially, and we present in Chapter 2 a report on the recent progress in the field, and identify tomorrow's challenges with these perspectives in mind.

The design of interactive and automated camera systems indeed covers quite a wide range of challenges. We propose the term of **computational virtual cinematography** to embrace all the numerical techniques related to interactive or automated cinematography in synthetic environments that build on computational approaches.

1.2 Aesthetic target tracking: problem definition

In this thesis we propose to study a specific sub problem of computational virtual cinematography: the **aesthetic tracking of targets**. In video games the camera tracking task consists in maintaining the main character on screen while ensuring other properties depending on the gameplay. For example in third person action games the camera manages to focus on objectives as enemies or interactive elements while always keeping

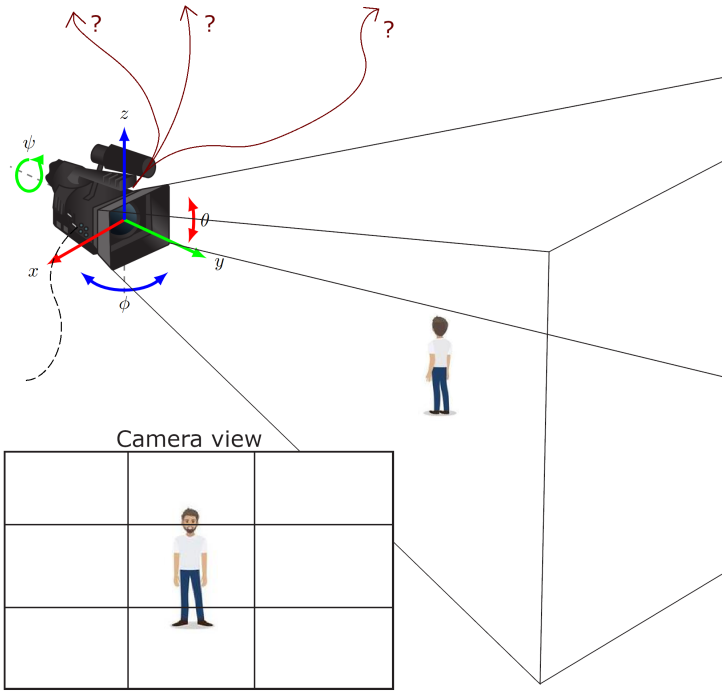


Figure 1.2 – Camera control problem for aesthetic target tracking: using the camera 6 degrees of freedom to maintain visibility on the target, ensuring an image composition and an aesthetic camera motion.

the main character on screen.

In competitive multiplayer games tournament (*i.e.* e-sport) or extensively, in sports that are broadcasted online or commented during live events, replays are often used to help the viewers to understand specific actions. Here, the tracking task has to convey the interesting events in an understandable and aesthetic or codified.

Multiple challenges arise when tackling this problem.

Challenge 1: visibility and collision The first challenge of camera tracking is to maintain the on-screen visibility of the target while avoiding collisions with the environment. A number of solutions exists to measure visibility (*e.g.* ray-casting, rasterization, visibility maps, ...) but it is still challenging to measure visibility **efficiently** in dynamic environments. Additionally, it is challenging to react to an occlusion: one needs to measure the visibility of the local neighborhood, and in some cases of the larger regions around the target. Choosing where the camera goes to avoid an occlusion is critical because in some cases the best solution is to be occluded for a short period of time to

reach an occlusion-free area instead of staying inside an area that will be occluded for a longer period.

Challenge 2: aesthetic dimension The aesthetics of a camera trajectory is carried by numerous properties (*i.e.* view angle, view distance, image composition) that are not compatible all the time (*e.g.* to avoid occlusion it will be difficult to keep a view angle or view distance). Moreover, at a higher level of expression, the notion of aesthetics is difficult to formalize in terms of metrics to optimize as it is subjective and dependent of the context offered by the content.

Challenge 3: dynamic dimension The dynamic nature of the environment that the camera goes through, adds a dimension of non-determinism in the target and occluders motions. This aspect of the problem makes difficult to plan on the long term the camera motion, target occlusion and an aesthetic camera motion.

Challenge 4: Coordination in real-time Finally, what's makes the overall task of aesthetic target tracking for interactive applications particularly challenging, is that each aspect mentioned above (*i.e.* visibility, aesthetic, dynamic) need to be evaluated and balanced in real-time.

The question we address in this thesis is therefore: "***How can we efficiently and automatically control a virtual camera to track one or multiple targets in a 3D dynamic environment ?***".

We present two works focusing on how to automatically control the camera (Figure 1.2) in order to enforce properties (*i.e.* collision, occlusion, view distance, view angle, camera trajectory jerkiness, framing, etc...) that are vectors to the cinematographic rules for virtual cinematography.

1.3 Publications

During this thesis we published two conference papers listed as follows:

Real-time Anticipation of Occlusions for Automated Camera Control in Toric Space (Conference Paper) Ludovic Burg, Christophe Lino, Marc Christie. In Computer Graphics Forum 39, 2020 [BLC20].

Real-Time Cinematic Tracking of Targets in Dynamic Environments (Conference Paper) Ludovic Burg, Christophe Lino, Marc Christie. In GI 2021-Graphics Interface conference, 2021 [BLC21].

1.4 Outline

In Chapter 2 we will provide an in depth view of the research field in virtual cinematography in which we show the recent advances in the fields.

Relying on our analysis of the virtual cinematography field, we present a first contribution in Chapter 3. We propose to rely on a local planning approaches (*cf.* section 2.4.2) to the problem of automatic camera collision and occlusion avoidance. The system relies on the toric space presented in chapter 2 and on rendering technique to achieve real-time performances. Despite its benefits, the system has limitations related to the local nature of the approach (*i.e.* the camera can get stuck in local minima).

To overcome these limitations we propose a second approach in the Chapter 4. On the same basis, we propose an automatic camera system that relies on a hybrid approach (*cf.* section 2.4.3). In this second approach, instead of using an existing parametric space (toric space) we introduce a new space, the **camera animation space**. We use GPU ray tracing to efficiently evaluate motions in the animation space with a custom trajectory quality metric.

In the chapter 5 we will conclude this thesis by summarizing the contributions and their limitations and talk about the perspectives of this research.

CHAPTER 2

STATE OF THE ART

In this first Chapter we are going to study the literature to see on which basis we can rely to address our problematic. We will be looking at the current state of the two relevant fields for this topic: (i) virtual cinematography and (ii) drone cinematography.

Since the seminal work of Jim Blinn in 1988 [Bli88], which searched for automated ways to position cameras in virtual environments, the area of interactive and automated virtual cinematography has addressed many challenging problems, borrowing and adapting techniques from the wide fields of computer vision, robotics and computer graphics, but also designing novel dedicated approaches.

The first survey related to virtual cinematography techniques [CON08] provided a general overview of the virtual camera control approaches and related applications. A more recent survey by Ronfard [Ron20] provides a historical perspective focused on film editing. The latter mainly covers the cinematic background and editing conventions, extracted from books written by real cinematographers and psycho-cognitive works studying this film grammar. The work reports contributions to the fully-automated computation of edits by using vision-based or early works on the transposition to virtual camera control and storytelling. However, these respective surveys do not cover the many recent contributions in controlling 3D cameras, in particular topics such as *movie previsualization* (the interactive prototyping of movies in virtual environment), or *aerial cinematography* (the cinematic control of unmanned aerial vehicles), which have emerged from the 2010s. In this chapter, we aim at extending these two surveys with an overview of both emerging applications and recent advancements in computational virtual cinematography. We also provide the reader with perspectives on the ongoing technical challenges and the remaining scientific locks. The surveyed contributions show that there is an increasingly strong convergence in the challenges tackled by the *virtual cinematography* (computer graphics) and *aerial cinematography* (robotics) communities.

We structured our approach by analyzing the evolution of camera control techniques as branches in a graph that displays both the practical use-cases, and the evolution in the use of underlying techniques as represented in Figure 2.1.

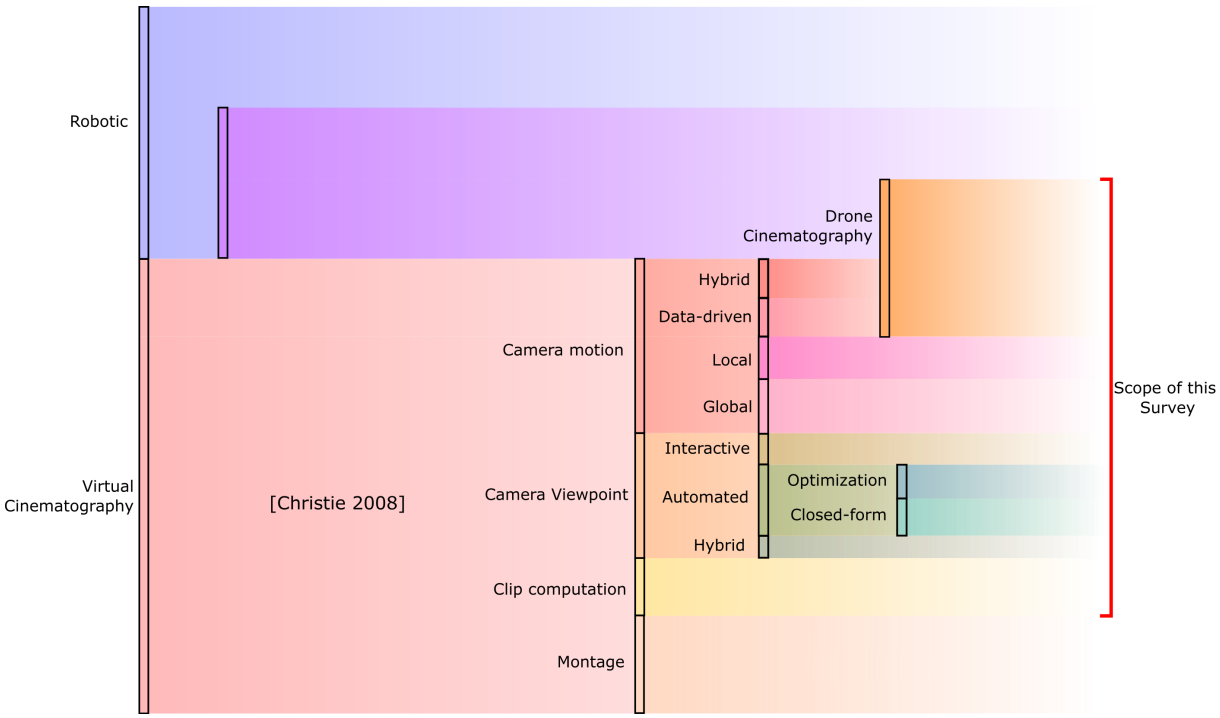


Figure 2.1 – Ranges of topics that will be addressed in this state-of-the-art

This chapter is split into three major sections related to advances in viewpoint computation, clip computation and trajectory computation. We open with the recent trends in the field and the open challenges to address.

2.1 Classification of recent techniques

There is a wide variety in computational cinematography techniques, which have been applied to a wide range of applications (*e.g.* computer games, drones, data visualization, multimedia). In order to structure the chapter, we propose three main criteria along which techniques can be sorted: (i) which problems are addressed, (ii) in which control space problems are expressed (*i.e.* how much indirection exist in the way we fix camera parameters) and (iii) which approach is used to solve them.

2.1.1 Classes of problems

Following what is done in the film and broadcast industries, we propose to distinguish four distinct classes of problems :

— **Camera placement problems** which consist in computing a camera configuration (assignment of intrinsic and extrinsic parameters) that ensures specified visual properties such as image composition (*e.g.* spatial layout of targets in the image, in/out focus on targets, relative angles to targets, but also properties driven by aesthetics such as balance in images or lighting).

— **Camera motion problems** which consist in computing a continuous sequence of camera configurations that ensures both visual properties over time (the same as those in camera placement) and also properties of the generated trajectories such as smoothness/jitter, similarity to standards (*e.g.* travelling, panoramic, dolly), but also higher-level properties such amount of information conveyed.

— **Camera cutting problems** which consist in deciding when are the best moments to switch between viewpoints and to choose to which camera to cut to, following elicited continuity editing rules.

— **Clip generation problems** which consist in selecting from a sequence of events occurring in the scene and from the characteristics of the scene, what are the most interesting parts to display, where the level of interest can be guided by different metrics (*e.g.* conveying actions, conveying only informative content, total duration of the clip).

In this overview, we do not consider the correlated problem of 3D layouts in scenes (*i.e.* how to place entities in the 3D scene to ensure expected visual and motion properties), nor do we address lighting issues. Both tremendously influence the problems we address, yet have received limited attention when addressed simultaneously with camera and clip computation problems. We refer the reader to notable exceptions like [ER07]; [Lou+20] for layouts and [Gal+18b] for lighting.

2.1.2 Classes of control space

Computational virtual cinematography intrinsically requires searching for the values of a parametric system (*e.g.* camera configuration, camera path, camera clip) by expressing relations between cinematographic properties and variables of the parametric system. These properties represent the *control space* solving constraints in both world-space (visibility, collisions, camera path smoothness) and in image-space (what the camera sees, and how: on-screen position, size, view angle of filmed objects). Camera space can be divided in four classes depending on the optimization scale:

— optimize on **low-level degrees of freedom**: *i.e.* position, orientation or control parameters for drones

— optimize **look-at/look-from**: camera position with look-at point position gives an indirect control on the camera orientation (Ranon 2014 [RU14], Joubert 2015 [Jou+15])

— direct solving from **image-space** constraints: on-screen points (Gleicher [GW92]), higher-level on-screen constraints (toric space [LC15])

— optimize in **mixed world/image space**: solving from a defined target on screen position and on world camera positioning constraints

2.1.3 Classes of approach

For camera control we can identify 5 classes of approach in the literature:

— **Fully interactive approaches** (full control to the user): direct control on degrees of freedom

— **Fully automated approaches** (full control to the machine, given user specifications):

  — Reactive approaches: *e.g.* visual servoing

  — Search-based approaches (local or global optimization): MPC, active contour, graph search, etc.

  — Close-form solutions: *e.g.* Blinn [Bli88], Lino & Christie 2012/2015 [LC12]; [LC15]

  — Data-driven / Machine learning-based approaches

— **Mixed-Initiative approaches** (knowledge to the machine, control to the user): shared decisions between user and machine (*e.g.* Director's Lens), by delegating low-level tasks to the machine, while providing the user with a fine control in higher-level (creative) tasks.

We can also consider classifying the approach based on their knowledge level:

— **explicit** knowledge models (procedural)

— **implicit** knowledge models (data-driven)

2.2 Camera Viewpoints

Placing cameras in 3D scenes is essential to any 3D application. In particular, how cameras are positioned in the scene will control how the scene content is conveyed (*e.g.* highlight or hide spatial and temporal relationships between objects). In turn, this will strongly impact the perception of the user. Consequently, cinematographers have long been elaborating a visual grammar of camera placements, enabling to properly convey 3D animated contents.

What we here call a viewpoint refers to two interconnected aspects. At a geometric level, a viewpoint corresponds to the set of values taken by each parameter of the camera (*e.g.* its position, orientation, focal length). At a semantic level, a viewpoint refers to the visual arrangement of objects on the screen (*e.g.* their location or size), which directly results from these camera parameters. However, the relationship between this semantics and the proper camera parameters is counter-intuitive for users, as well as highly nonlinear mathematically speaking. How to intuitively interact with or compute camera parameters, to satisfy a desired viewpoint semantics, still remains a central challenge in 3D cinematography.

Controlling a virtual camera in a complex 3D scene remains a tedious and technical task. The community has been trying to design innovative interaction or computation techniques, to make the exploration and inspection of 3D contents easier. We here do not provide a complete survey of 3D navigation techniques, which can be found in [JH13]. Rather, we list a few noteworthy contributions, which are laying the ground for camera control techniques at large, and aim to provide hints on how they can influence current or future 3D cinematography techniques.

2.2.1 Interactive methods (viewpoint manipulation)

Viewpoint manipulation techniques aim at providing the user with a means to interactively update camera parameters, *e.g.* to inspect 3D objects or navigate through complex 3D scenes. However, as explained in [CON08], in these tasks, the principal challenge remains to provide interesting control metaphors (*i.e.* mappings between an input device and the camera parameters).

Most early works, discussed in [CON08], have provided a limited solution to the interactive control of cameras. Their proposition were limited to control metaphors defined at a strictly geometric level. Indeed, they proposed direct mappings between 2D

mouse inputs and the 7 camera parameters, or moving the camera onto user-defined surfaces. In the case of proximal inspection, early works have proposed to use potential fields [HA+92] or vector fields to avoid collisions. They have also proposed dedicated techniques to handle cavities and sharp turns in the scene geometry. Potential fields appear early in the robotic fields for path planner and have been used in the virtual cinematography community. They are derivable functions defined locally around the camera and map the properties to minimize. By this mean it is possible to find a path from the current position to a local minimum of the field.

Due to their lack, more recent works have been seeking to provide more intuitive and effective control metaphors, aiming to better consider the semantics of controlled viewpoints.

Sudarsanam *et al.* [SGS04]; [SGS09] proposed to control the scene perspective based on the concept of vanishing points, which artists are very familiar with when sketching storyboards. Their *IBar* and *CubeCam* interfaces build on this same concept. Both propose to visualize potential camera operations, relative to what the camera currently sees, through a cube widget centered on a focused object in the camera view. In *CubeCam*, this cube widget represent the camera perspective at three different levels (1-pt, 2-pt, or 3-pt perspective), depending on how many vanishing points are considered. While manipulating this widget at a given perspective level, the user can perform a subset of camera operations (*e.g.* pan, zoom, zoom + dolly, camera translations or rotations, change of focal length) around the selected object. Each operation is linked to the update of extrinsic, intrinsic, or a combination of camera parameters. This in turn results in changes in the rendered image of the scene. They also propose to save camera views as *bookmarks*. Their system can then display nearby bookmarks around the cube widget, which allows the user to select them as a new starting camera view. They finally propose a ghosting mechanism as a way to visualize how a camera operation may impact the camera view. While performing camera operation in ghosting mode, the system superimposes the image rendered from the controlled view on top of the initial camera view. This helps users in first exploring how camera operations impact on the rendering, before actually applying them.

One remarkable early proximal camera technique is the point-of-interest (POI) technique proposed by Mackinlay *et al.* [MCR90]. POI consists in first selecting a point of interest (*i.e.* target object). Camera motions can then be performed forward or backward in the direction of this POI, using a logarithmic speed depending on its distance to

the camera. The camera can also rotate around this target object, with constant angular speed, thus providing orbiting motions at faster speed at high distance, and slower speed at low distance. Following the POI concept, Moerman [MMG12] proposed Drag'n go, where the POI is computed with a ray cast from a pixel clicked by the user. The camera then moves along a straight path (*i.e.* the launched ray), while the mouse cursor controls the amount of camera motion along this path. The user can also control the amount of rotation through the x component of the mouse position. The on-screen position of the POI is also enforced during all camera motions.

Though efficient and simple, the POI technique and proposed improvements were lacking generic solutions to handle cavities and sharp turns when traveling 3D scenes or inspecting complex 3D geometries. Hence, further research has been devoted to the proposition of generic solutions able to handle such specific cases. Among prominent works, Boubekeur [Bou14] proposed *ShellCam*, a geometry-aware proximal inspection technique. Their idea is to leverage the rasterization pipeline to determine an implicit surface (*i.e.* an offset *shell*) around the visible geometry of an inspected object. Given a current camera position and orientation, the local tangent to the shell is determined from the averaged visible normal. This allows for smooth rotational movements around objects, regardless of the complexity of their geometry, while rotating the camera to always point at the object. By leveraging the depth buffer, they also approximate the distance to the visible geometry. This naturally provides logarithmic translation motions. The main advantage of their technique is the implicit definition of a multiscale set of shells (*i.e.* a shell can be defined at any distance to the geometry). As well, rendering the visible geometry is more detailed at a closer distance. Hence, close shells will more closely follow cavities. Symmetrically, shells become smoother and smoother at an increasing distance, hence allow smooth orbiting motions around the object.

McCrae [McC+09] also leverage GPU hardware computation. They proposed to encode the local environment around the camera into a cubemap, with knowledge of both normal, up vector and distance to geometry. They propose to use a level-of-detail strategy, by abstracting an object with simple primitives when it is far, while directly using its detailed geometry when closer. Then, the authors proposed to use the distance to geometry to adapt the camera speed and viewing frustum (*i.e.* the near/far plane distance) to the navigation scale. And, more importantly, they use it to smoothly handle collision detection while avoiding costly geometrical computations. They instead leverage the distances in the cubemap to compute a repulsion force, by comparing them to a

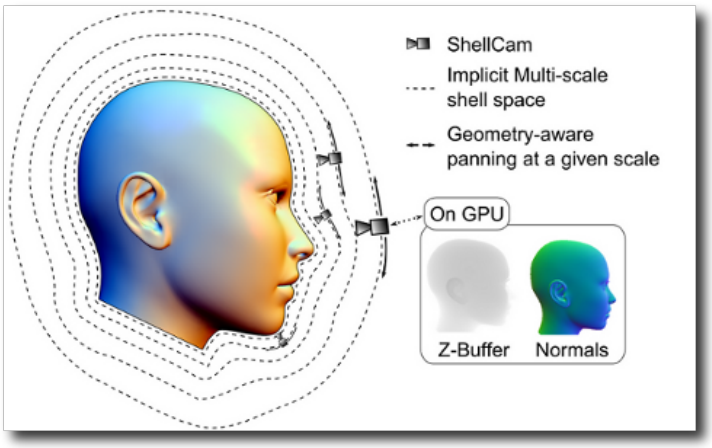


Figure 2.2 – ShellCam [Bou14]

distance threshold (*i.e.* a sphere around the camera). This enables generating collision-free fly-through paths to move the camera forward (or backward) to a new scale level (*e.g.* at world-scale), or camera motions within a fine detailed scale level (*e.g.* within a city or building). They finally propose to detect when the camera is stuck in a cluttered area, and to consequently decrease the distance threshold.

Marton *et al.* [Mar+14] proposed *IsoCam*, which also relies on the concept of moving the camera on a multiscale set of implicit iso-surfaces around the inspected object. Note that, the computation of these iso-surfaces here relies on a hierarchy of simplified geometries, from the actual object's geometry to rough meshes. Authors further propose to leverage an idea similar to design galleries [Mar+97]. In the image rendered from the current viewpoint, similar viewpoints are also displayed. They are automatically extracted from a database of pre-computed views around the object of focus, using a metric combining the similarities of rendered images and of camera configurations (in position and viewing direction). They can be clicked to teleport the camera to the associated camera configuration.

Hachet *et al.* [Hac+08] derive an extension from the POI technique. They first propose to use sketching to select a region of interest (ROI), instead of a single POI, which provides a more robust control on where the user wants to focus. They then design a new 3D widget, called *Navidget*, which allows a better control on the positioning and view direction of the camera around this ROI. *Navidget* is a sphere centered on the central POI. They propose several strategies to initiate the radius of this sphere, among which the lowest distance to geometry enables to find camera positions from which the

whole ROI might be seen. They finally propose to control the camera viewing angle (*i.e.* its position and viewing distance) by simply pointing to some location onto the visible half-sphere surface of *Navidget*. The camera is consequently moved to a position located along the corresponding normal, and at a distance equal to the sphere radius. In turn, the camera's viewing direction is computed to always point toward the central POI.

All these interactive techniques have been mainly proposed for the exploration of 3D scenes, and the proximal or distant inspection of objects. Often, they are even limited to a single object of focus, or to manipulating low-level geometrical concepts or primitives. But more importantly, such technique miss knowledge on how cameras are controlled in the real world (*e.g.* in cinema or documentary) to convey real world situations. This may lead to unnatural 3D camera views and motions. Better conveying virtual worlds through the lens of cameras intrinsically requires manipulating high-level concepts such as the visual layout of objects (*i.e.* how they are visually arranged on screen). Hence, using available camera control metaphors offered in current 3D software designing the aesthetic of rendered images remains a highly technical, artistic, and time-consuming task.

2.2.2 Automated methods (viewpoint computation)

In more and more 3D application (*e.g.* virtual tours, cultural heritage, entertainment), designers wants to improve the immersion of users in 3D scenes, through a more narrative experience. The pressing requirement to control the aesthetic of camera viewpoints has then pushed researchers to propose techniques to automate the placement of cameras. This automation problem is also known as a *viewpoint computation* or a *camera composition* problem. Visual composition criteria generally accounted for in the literature comprise the on-screen positioning or ordering of objects (*e.g.* to the left, right, top, bottom), their size (*e.g.* using a cinematic scale, such as which proportions of a character should be included in the image), or view angle (*e.g.* from front or back, from a side, from above or below). A better insight into such criteria can be found in cinematography cook books [Ari91]; [Mas98]; [TB09b].

The specification of these visual composition criteria is generally performed through a declarative process. Early works [CON08] allowed user to declare such criteria as constraints on the camera parameters. Automated systems then generally cast the camera

placement problem as a constraint solving problem (CSP) or an optimization problem. In the former constraints are translated into a set of equations and inequations to solve. In the latter, they are translated into a set of cost functions (to minimize) or satisfaction functions (to maximize) taking camera parameters as input.

Notable early works comprised Blinn's "Where am I? What am I looking at?" [Bli88], as well as Gleicher and Witkin's "Through-the-lens camera control" [GW92]. Both proposed to control the screen position of selected objects and/or their distance to the camera. Blinn's solution is based on the inversion of the camera projection matrix, and on the resolution of the camera position and its orientation in two separate steps, which his system iterates on. Gleicher and Witkin instead proposed to cast the problem as an inverse kinematic problem on camera parameters, where the minimized costs are linked to the on-screen projection errors of the controlled points in the scene.

Recent state-of-the-art techniques use either close-form solutions, when possible, or optimization techniques. Although some early approaches have relied on constraint-solving (CSP) techniques, to our best knowledge recent approaches no longer use them. Indeed, computing camera viewpoints is essentially an inverse problem involving constraints that remain difficult to combine. As a result, such problems are most often under constrained or over constrained. In this context, optimization provides more flexibility on how constraints are combined, while making it possible to find good approximate solutions when a close-form solution cannot be found. Some recent approaches also combine the advantages of close-form and optimization-based solutions.

Most works based on optimization use the same form:

$$\arg\min_{x} E(x(t), c(t)) \tag{2.1}$$

With E the energy combination of the properties to evaluates, $x(t)$ the camera configuration over time and $c(t)$ the context in which the camera is (*i.e.* the scene topology, the environment, the dynamic objects, etc...).

Optimization on camera parameters

Most proposed techniques rely on the formalization of viewpoint constraints (visual composition rules) as cost functions (or satisfaction functions), and their aggregation into a fitness function built as a weighted sum of costs (or satisfactions). They then rely on the search, in the 7-dimensional camera parameter space, of a camera configuration

that minimizes costs (or maximizes satisfactions).

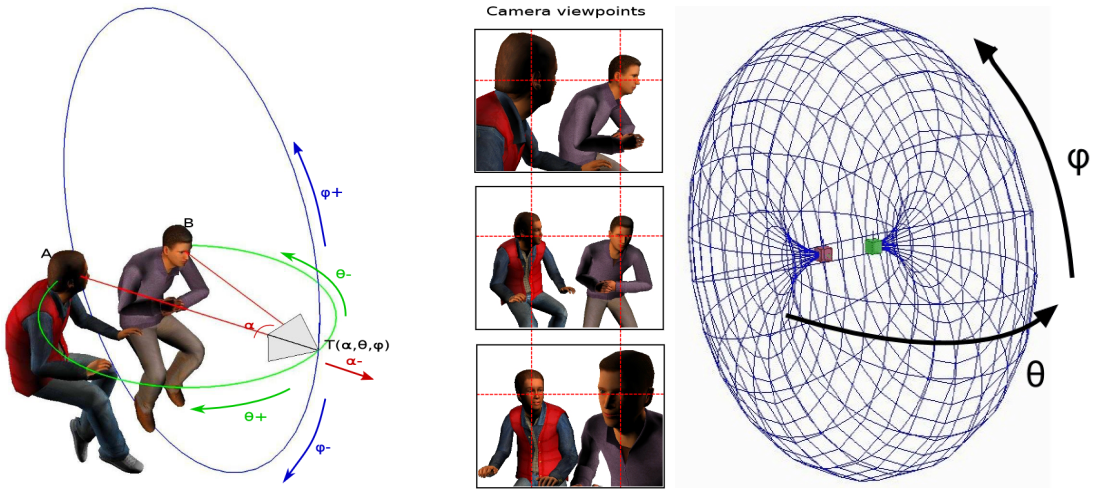
Ranon and Urli [RU14] propose a very general and efficient solution, which relies on a particle-swarm optimization (PSO) technique, and on a smart initialization procedure.

One might need to make a trade-off between the computational cost and the accuracy in the evaluation of the visual constraints' satisfaction. Ranon *et al.* [RCU10] proposed a simple language to define camera constraints and a way to accurately evaluate the evaluation precision within a defined computation time constraint.

Some works focused more on composition features as Abdullah *et al.* [Abd+11] encoded multiples standard cinematography rules and use a PSO (Particle Swarm Optimization) in order to satisfy the constraints. Sanokho *et al.* [SC14] focuses on visual balance. By using semi-annotated data, they are able to correct the visual balance of synthetic shot. First, features are extracted by hand from multiple shots with one and two actors from which a feature space is created. Secondly, the center of each cluster of data is considered as a good balancing point. Finally, from a synthetic shot the balance is evaluated and corrected from the optimized space.

Close-form solutions

Historically, the first attempt to provide a closed-form solution was proposed in [Bli88]. Blinn mention a viewpoint computation challenge he has been giving his students for a decade. Considering a planet and a ship, with the planet being at a given distance and its axis being vertical on screen, find a camera configuration that satisfies input screen positions. His solution relies on inverting the camera projection matrix to solve the camera position and the camera orientation in two separate steps. He then proposed to iterate on solving each sub problem, while the other is considered fixed, until convergence. More recently, Lino and Christie [LC12] proposed a low-dimensional search space, called *Toric manifold* (Figure 2.3). They first observed that given the exact on-screen positions of two targets, solutions camera positions belong to a spindle torus around them, and that the camera orientation is also fully determined by a picked camera position on this torus. Solution camera positions can also be parameterized with two angles: a vertical angle and a horizontal angle on the torus. The key provision of their toric manifold is that one can now intrinsically encode a key visual feature (the on-screen positions of two objects) into a 2-parametric search space. Leveraging Toric space's nice properties, they have been able to propose the first fully algebraic solution to Blinn's problem.



(a) Camera viewpoints for different toric parameters (b) The toric manifold

Figure 2.3 – The toric space [LC15]: a camera position representation dedicated to framing

Optimization in intermediate search spaces

In [LC12], Lino and Christie also proposed to leverage their close-form solution (the Toric space) and provide a nice extension to the on-screen positioning of more than two target objects. They first propose to fix the on-screen positions of a pair of target (*e.g.* two protagonist characters). Then, they cast the problem as searching a 2-parametric camera position onto the torus associated to this pair of objects, while minimizing the on-screen projection error for other objects. Note that for three objects the on-screen positioning is well-constrained (there exist only two discrete solutions), while for more objects the problem is over constrained, hence requiring either a trade-off between constraints or a relaxation of some constraints.

2.2.3 Hybrid methods

The rise of better interaction metaphors and procedural generation models have also enabled the emergence of a new trend in the computer graphics community: the proposition of mixed-initiative content creation tools. As for camera control tasks, the main idea behind have been to try combining the power of automated systems and the creativity of human artists. In this quest, one important factor is to propose automated systems which can compute camera viewpoints in interactive time, while providing users with

interaction metaphors providing a fine and effective control of the visual result.

Being able to propose effective control techniques would indeed enable the user and the system to share the burden of solving the camera control problem together, by letting the user have the control artistic decisions while the system would make the technical and tedious tasks as transparent as possible to the user.

Lino *et al.* [Lin+11b] proposed Director's Lens, a system assisting the user in the interactive exploration of available viewpoints in the scene. They rely on an idea similar to intelligent gallery [Vie+09]. Viewpoints are arranged in a grid, which the user can browse. Their system then allows the user to select a viewpoint, and to virtually dig into the viewpoint, by using a 6DOF physical device tracked in position and orientation, reproducing the way real cameras are controlled. This allows a rapid exploration of semantically meaningful viewpoints, while also providing users with a fine and intuitive control of the final viewpoint.

More recently, Lino and Christie [LC15] have proposed an intuitive and robust manipulation of a larger set of visual features (on-screen position, size or view angle) on a pair of target objects. Their solution relies on direct algebraic computations in Toric space, an intermediate camera parameter space they proposed in [LC12].

As a conclusion to this section, to our opinion the viewpoint computation problem is mostly solved for specific cases (*e.g.* simple geometrical constraints in world space or screen space, or for up to two or three objects). As a remaining challenge, there is now a pressing need to account for higher-level aesthetic rules (*e.g.* image balance, or quality in composition) and to address more complex problems (*e.g.* considering additional camera parameters such as the depth of field, or moving to stereoscopic cameras).

2.3 Clip computation of 3D animation

The idea of processing automatically an animation sequence from a defined 3D animation is a natural though. Indeed, when the animation of the scene is done by an animator, placing camera is a tedious and time-consuming task. Moreover, besides computing clips from a finished animation, clip computation includes summarization of a filmed sequence (*e.g.* in game replay for online multiplayer games) that keep the most relevant parts of a sequence. The main questions to answer here are: (i) How to detect and define "interesting" events of the sequence ? (ii) How to handle uninteresting parts of the sequence ? (iii) How to convey interesting parts ?

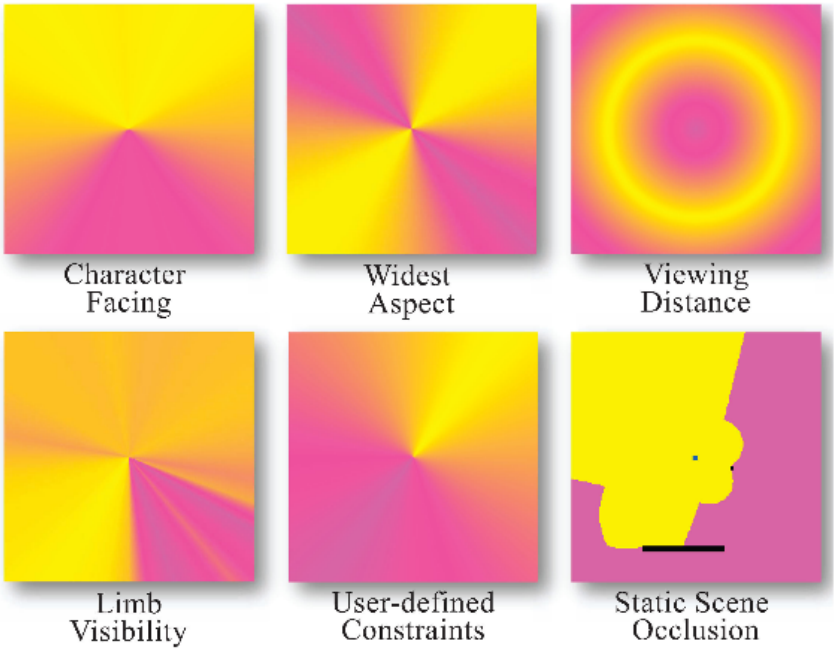


Figure 2.4 – Decomposition of potential field function constraints [Yeh+11a]

Assa *et al.* [Ass+08] proposed a work on maximizing the viewpoint entropy of a character animation. The viewpoint quality is measured as a combination of generic descriptor and pose specific ones. The combined descriptor is represented as a potential field. To generate the camera motion an optimization phase is performed to minimize an energy function for the camera displacement on the potential field 2.1 with $c(t)$ the character animation from which the potential field is generated. The energy function in separated with the internal energy on one side and the external energy on the other side. The external energy refers to the score of the camera on the potential field and the internal energy refers to the cost of moving the camera in a specific spot. In the same way, Assa *et al.* [AWC10] proposed an offline clip generator system of an animated character. First, according to cinematographic guidelines, multiple cameras are positioned on the scene. Then, the character motions data is analyzed using signal processing techniques and by a correlation measurement, the best camera to use can be found and set as the active one. Using an idea similar to this viewpoint entropy, Yeh *et al.* [Yeh+11a] proposed an offline camera path planner targeting to maximize the shot quality on a character animation by computing a potential field from multiple constraints (figure 2.4). Yeh *et al.* [Yeh+11b] then extended this approach to an offline clip generation for multiple characters. By analyzing the characters animations, they detect

important events (*e.g.* when characters cross ?), which should be conveyed in the clip. The sequence is analyzed through the targets localization and their animation data. When a lot of activity is detected, it becomes an event to include in the clip. Using the sequence decomposition, an optimization step is performed to maximize the viewpoint entropy of each event and the transitions between the event clip are flags suggesting the cuts.

The same idea as [Yeh+11b] but with a local approach Galvane *et al.* [Gal+13] proposed to use steering techniques, first proposed for characters [Rey99], using steering forces to attract the camera to the point of interest of the scene. The scene is analyzed using flying camera. Thus, events are found this way and the camera is attracted to these events using forces. The framing is handled using a modified toric space that add a security space to avoid collisions between the camera and the targets.

Xie *et al.* [Xie+18] proposed a trip synopsis system by building trajectories by pieces. Their goal is to film landmarks with drones. They build two kinds of trajectories, the landmark's exploration trajectories that are meant to optimize the view on the landmarks and the transitions between the landmarks areas. The landmark exploration trajectories are built using the landmark's silhouette with potential fields and a quality view metrics based on the visual entropy principle. The view points are positioned according to the minimization of the potential field of a landmark. For each view point the landmark is rendered to compute the viewpoint entropy score. Then, the interest points are encoded directly inside a saliency texture for each landmark. Finally, 5th degree B-spline is generated between the better connected view-points. For the transitions trajectories, a graph is built to map the landmark position. Then, the transition between the landmark is solved using a Traveling Salesman Problem where the cost of a transitions account for collision and visibility.

Huang *et al.* [Hua+16] proposed a trip synopsis system. The goal of the system is to make a 60 seconds clip that summarize a drone trip. To accomplish that, optical flow is analyzed. Indeed, optical gives a good idea of what is happening on screen, the higher the optical flow magnitude is the more event are happening on screen.

2.4 Camera motion

Computing static viewpoints is almost a solved problem. However, the addition of the temporal dimension to this question raises new issues. For example, to be able to

interact with the 3D environment in real-time, the camera computation time become crucial. The addition of the temporal component also raises the question of the dynamics of the scenes and how to handle it. Moreover, planing the motion of the camera imply that we need to anticipate the future state of the scene and the targets' behavior.

Proposed approaches to generate camera motions can be divided in four classes: (i) global approaches: the camera is aware of the entire scene topology. It has the benefits to be accurate and avoid local minima in the scene but is usually sensible to dynamic environment due to the heavy computation costs of the data structures update used for the topology representation (ii) local approaches: the camera only have a partial knowledge of its surroundings. It presents the opposing advantages and disadvantages of global approaches, it is usually low cost and can react easily to dynamic changes on the scene but have trouble to get out of local minima (iii) hybrid approaches: trying to convey benefits of both global and local approaches (iv) finally, data-driven approaches usually relying on deep learning tools or the optimization of a latent space based on large amount of data.

Trajectory evaluation is critical to define the quality of the result. This is the reason why, in the last part of this section we analyze the different way the community evaluate camera trajectories. To evaluate camera trajectories most of the works either conduct user evaluation or compute satisfaction scores by evaluating criteria on the camera, we can split these criteria in four categories:

— The relation between the camera and the environment: the camera collision and the obstruction of the camera visibility by the environment

— The camera on screen properties: what does the camera has to see and where are the targets on screen

— The camera motion: camera speed, jerkiness of the trajectory

— The cinematography: the on-screen motion properties

2.4.1 Global planning

Global planner approach usually relies on well known technologies and techniques borrow to the AI and robotic fields as roadmaps, cell decomposition, PRM [Yan+17], RRT and other optimization tools like potential fields [HA+92]. These data structures are then, fill with the scene topology information and updated in runtime.

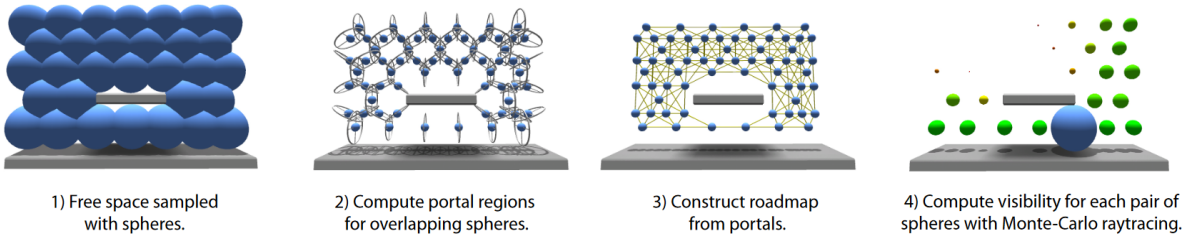


Figure 2.5 – Visibility graph and road map computation [Osk+09]

Early, data structure like graphs were commonly used to encode the properties related to the scene topology. Geraerts *et al.* [Ger09] proposed the corridor map method. It consists in two phases, an off-line construction phase and an online query phase. In the construction phase, the environment is analyzed and mapped using a graph as an environment representation. The graph construction is based on Voronoi diagram, this allows to identify the principle navigable axis of the environment. Then, sampling these axes and assigning to each sample the radius of the largest empty disk around it, permit to define the cleared area. This data structure define what the authors call the corridor map. In the query phase, the data structure is used to ensure that the camera is collision free and by using steering forces the camera is attracted by the target and an ideal camera track.

Oskam *et al.* [Osk+09] proposed a real-time visibility-aware camera path planner to follow a target. In a pre-computation step, a roadmap that stores the visibility and collision information is computed (figure 2.5). First, collision-free spheres are sampled in the scene. Then the surfaces created by the sphere superposition (*i.e.* represented as disks) are considered as nodes and all nodes computed in a sphere are connected by edges in the roadmap. Finally, the roadmap is augmented with visibility information by performing a Monte-Carlo sampling between all pairs of spheres. At runtime, given a start position, goal position and a focus point, the system computes the shortest collision-free path using an A* algorithm. The cost of a transition accounts for the length of a transition and the visibility toward the focus position. Following the same idea, Lino *et al.* [Lin+10]; [Lin13] proposed a global virtual camera planner accounting for visibility and viewpoint semantic. For this purpose, the director volumes are introduced which are extension of the semantic space portioning of Christie *et al.* [CN05]. In a pre-computation step, a 2.5D cell-and-portal partitioning is performed on the scene. At run time, two partitioning are performed for every cell: a visibility partitioning and

a semantic partitioning. These two partitioning generate respectively volumes encoding each information. Directors volumes are computed from the intersection of the two partitions. The roadmap is then generated by sampling the portals and shared edges of the director volumes. Finally, an A* algorithm is performed to search in the graph accounting for semantic and visibility. As the target moves, the volumes and the resulting roadmap are recomputed.

On another note, manually placing camera rails and static cameras in a 3D environment is tedious and time-consuming for designers. Jovane *et al.* [JLC20] proposed a virtual camera system that automatically generates cameras according to the scene geometry. In a pre-computation step, the navigation mesh of the scene is analyzed and a dense collection of potential cameras is generated from a skeleton representation of the navigation mesh. Then a RANSAC driven clustering is performed to generate camera rails which are then organized in a graph representation. To reduce computation at run time, additional visibility information is also added to the graph similarly as [Osk+09]. At runtime, using a physical model, the camera is attracted to an ideal position that is computed to be the best camera position for the current frame. The pre-computation step has reduced the research space to a set of feasible camera rails. The ideal camera position is computed by an optimization step within this reduced research step, simply enforcing the framing and the coherency of the camera motion (*i.e.* avoiding jerkiness).

More recently, the drone cinematography field raised a lot of attention. Joubert *et al.* [Jou+15] proposed to use classical spline-based animation techniques on camera parameters (look-at, look-from) to design virtual motions, which they apply to unmanned aerial vehicles (UAV). However, the design of camera animation with cubic splines remains a tedious and time-consuming process. Further, drone control space is very different (*e.g.* rotor speed) imposing new constraints, such as the feasibility of trajectories. On this matter, Gebhardt *et al.* [Geb+16] proposed an optimization-based framework for trajectory computation from a set of key-frame to a feasible trajectory for a given motion model (*e.g.* quadrotor, robot). In [GSH18] a feasible trajectory for a drone is optimized using key-frames set by non-expert user. The trajectory is constructed as smooth as possible over time in order to minimize jerkiness. Roberts *et al.* [RH16] proposed an optimization algorithm that take as input a non-feasible drone trajectory designed by a non-expert user defined by some key-frame and return a feasible one. The progress curve of the robot that match its displacement model is firstly solve by a close form solution for the speed, acceleration and jerk. Then this progress curve is optimized

trying to match as closely as possible to the user specified progression curve.

Global approaches work well on mostly static scene as these approaches need to pre-compute huge data structure. Moreover, the cost of updating it in runtime can be expensive and not adapted to highly dynamic scenes. Local planning approaches is generally used to address dynamic environments.

2.4.2 Local planning

Local approaches are usually low budget in computation time and memory. Moreover, they usually address dynamic scene elements naturally by their reactive design. However, they are strongly prone to fall into local minima due to their local knowledge of the environment.

Steering based motion are commonly used for local approaches. By looking only for the necessary information to compute the camera position for the next frame the system does not require pre-computation and can react in case of dynamic changes on the scene. Burelli *et al.* [BJ09] proposed a force based system that rely on potential fields. The idea is to map the local environment around the target with functions. By following the slope of these composed functions they are able to define forces that will move the camera position and look-at. They use the visibility as a float instead of an integer, this mean they can specify how much of a target to see. The second function is the projection size of the target on the image. The last function is the view angle. Each function gives a force that attracts the camera and the final displacement force is computed through a weighed sum of the 3 functions. Moreover, following the same idea as [Yeh+11b] but with a local approach, Galvane *et al.* [Gal+13] proposed to use steering techniques, first proposed for characters by Reynolds [Rey99], using steering forces to attract the camera to the point of interest of the scene. The scene is analyzed using flying camera. Thus, events are found this way and the camera is attracted to these events using forces. The framing is handled using a modified toric space that add a security space to avoid collisions between the camera and the targets.

Visibility and collision are expensive to compute because of the high dimension problem of camera displacement. This is why, for more efficiencies some work focuses on extracting local information by using GPU computation and rendering techniques.

Christie *et al.* [CNO12] proposed to render a prism around the camera position using the targets as sources. Doing so the visibility can directly be extract inside the

prism. Thus, without heavy computation time, visibility is evaluated for a lot of potential camera configuration even if it is restrained around the current camera position.

Argelaguet *et al.* [AA10] proposed to optimize the speed of a camera from a camera track defined by a set of key frames. The optimization step accounts for five criteria: (i) improving the camera animation smoothness that is a standard guideline for camera animation; (ii) minimizing fatigue that can cause different symptoms (*e.g.* motions sickness, headache); (iii) maximizing the information given by the sequence; (iv) creating an interesting sequence, meaning accelerating when the on-screen information has already been seen or are not interesting; (v) generating a concise sequence *i.e.* that has a length adapted to the scene. Habituation maps are introduced and combined with optical flow maps and image saliency maps to perform the optimization. The concept of habituation refers to the degree of novelty of objects in the 3D scene and helps to maintain the users' attention.

Local planner are able to handle dynamic scene but have hard time to get out of local minima. Thus, we can imagine solution that combine local approaches and global approaches to get the benefits of the two approaches.

2.4.3 Hybrid planning

The core motives behind hybrid approaches is the will to create efficient techniques for application that have hard time constraints while having better trajectory results than local approaches.

Li and Cheng [LC08] proposed an automatic tracking system for one target. The objective is for the camera to keep a defined view angle on the target and avoid occlusion. For the path planning part, a lazy PRM is used [BK00] allowing the system to map the environment as a graph. The graph is build around the target and updated in real-time at runtime. The evaluation of the nodes and transitions inside the graph is computed using the difference between the ideal viewpoint and the current node position.

Burelli *et al.* [BY10a] proposed CamOn, a hybrid automatic virtual camera path planner. The idea of this approach is to first optimize locally to compute an ideal camera pose using a potential field accounting for image composition; if the local search fails, a global search is performed accounting for collision and visibility using an Evolutionary Algorithm [BY10b]. Finally, a path is computed between the current camera position and the ideal camera position using a PRM.

One of the most used techniques to create hybrid approaches is the Model Predictive Control (MPC). The idea of MPC is to find the next state of the system while accounting for the future within a time horizon. In the case of camera path planning it allows avoiding local minima by anticipating the system state inside a time horizon by knowing only the current state of it. To account for dynamic changes, the anticipation is recomputed at a higher frequency.

Litteneker *et al.* [LT17] proposed an automatic camera system that computes an optimized camera trajectory for scripted and unscripted scenario. Their optimization process relies on two energies similarly as [Ass+08] presented in section 2.3, the optimized energy is split in an internal energy encoding visual properties (visibility, shot size, relative angles, rule of third and look space) and an external energy encoding the camera path smoothness. The energy optimized in 2.1 is then:

$$E(x(t)) = E_{int}(x(t)) + E_{ext}(x(t))$$

Two successive optimization steps are applied. First, the keys frames are optimized through a simulated annealing only accounting for the internal energy. In a way similar to an active contour model, they secondly re-optimize key frames through a gradient descent taking into account both energies. In case of a scripted scenario, the optimized trajectory can be computed offline. In case of an unscripted scenario they use an MPC strategy to enable its online computation. They rely on a Taylor series to anticipate future key frames along the trajectory, from past key frames. They then run their 2-step optimization process on this anticipated trajectory, and steer the camera toward the first key frame. Nageli *et al.* [Näg+17a] proposed a real-time trajectory generator for aerial cinematography. The drone trajectory is optimized accounting for visual properties on multiple targets using MPC. Properties include: image space positions, projection size, relative viewing angle, desired camera pose and visibility. Moreover, they add a constraint to their system to avoid collisions with the environment and dynamic objects in the scene by using a potential field representation and a hard constraint. In [Näg+17c], the system is extended to automated multi-drone cinematography in cluttered and dynamic environments. Using MPCC the drones positions are planned in parallel along a time horizon. The targets positions are anticipated using a Kalmann filter and input to the drones planning system along with the framing input set by the user. The drones trajectories are constructed by optimization and account for mutual visibility issues. First

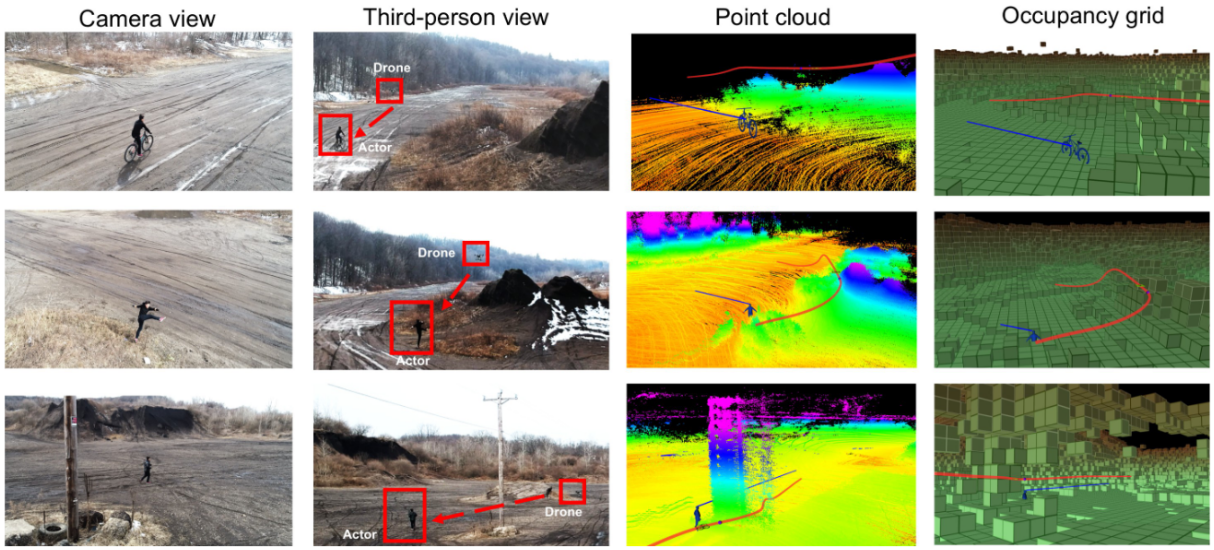


Figure 2.6 – System setup and results on different situations. [Bon+19]

a virtual camera rail is specified by the user. The system applies the local MPCC optimization to minimize the difference between the drone position and the virtual camera rail while avoiding dynamic colliders.

Bonatti *et al.* proposed a drone cinematography system in which their quadrotor is able to reconstruct the environment surrounding the drone as an occupancy grid. In [Bon+18]; [Bon+19], a 3-step planner was proposed for a single quadrotor (figure 2.6). First, the target is tracked using a vision-based detection neural network. Then, the target displacement is anticipated using a Kalman filter in a way that the targets' trajectory can be forecast in the 3D environment. This anticipation helps to build the "ideal" trajectory for the drone, "ideal" meaning maintaining the drone relative position (*i.e.* distance, relative angle) to the target overtime. Finally, the real drone trajectory is optimized for (i) Avoiding collision and occlusion with the environment, (ii) Staying as close as possible to their ideal trajectory and (iii) Avoiding jerky motions. This system is extended in [BBS20] to support multiple automatic drone planning with montage between the different shots. Three criteria are added to the cost function (i) Avoiding drones to see each other, (ii) Trying to have as much viewpoint diversity as possible between the drones and (iii) Avoiding user-defined viewpoints area.

Galvane *et al.* [Gal+18a] proposed a multi drone cinematography system based on the drone toric space (DTS). The DTS is a modified toric space representation of Lino *et al.* [LC15], in which a security radius is added around the targets to ensure a min-

imal security distance between the drone and the targets. The system allows the user to control the drone either through a standard joystick controller or designing by hand a rough trajectory that will be optimized to ensure collision-free and C4 continuity trajectories. The user can also specify the framing using through-the-lens interaction. For the collision-free path computation a road map similar to Oskam [Osk+09] is generated and encodes target visibility. Then, considering a sketched trajectory, the system searches for the closest trajectory available in the road map and similarly to [Geb+16] the found trajectory is optimized to be C4-continuous using a polynomial representation. In runtime, the trajectory is recomputed at a higher frequency to ensure it's still not collided. Finally, to handle multiple drones, a min conflict optimization is performed to avoid mutual drone visibility and ensure framing requirements.

Sanokho *et al.* [San+14] proposed to create a collection of extracted camera track from films to use them on virtual environments. The transitions between the tracks (*i.e.* cuts or smooth transition) are pre-computed and defines the camera motion graph. At run time, the visibility is checked by voxelizing the close future path and by launching rays between the voxels and the targets. If none of the voxel are visible the system use a transition toward another track. Otherwise, a path is built using the visible voxel. Each camera trajectory is stored as a series of camera configurations in toric space coordinates around the moving targets.

Most of the works focuses on how to produce a "good" camera trajectories with more or less time constraints. However, the definition of a good trajectory is still not well define. One of the possible solution to go around this, is to use data driven techniques to learn trajectories.

2.4.4 Data-driven camera motions

Led by the motive to produce trajectory similar to the one produced by human operator (*i.e.* for film or animated films), we can turn the problem over and produce camera trajectory from examples extracted from films.

Typically, Sanokho *et al.* [San+14] extract films camera trajectories using estimation algorithms, and expressing them in the local basis of a character, in order to build a camera motion graph (see Section 2.4.1 for more details). In the same idea, considering that details is an element of style for camera motion, Kurz *et al.* [Kur+14] proposed a style transfer system based on camera trajectories extracted from movie clips. First,

a database is built by extracting the trajectories using camera motion estimation algorithms (Structure-from-Motion [Özy+17], visual SLAM [TUI17]) and manually labeling the trajectories according to their style. Then, each trajectory is decomposed by separating the shape and details. The shape is computed using a Taubin filter and the details are extracted using a local frequency analysis (Gabor decomposition). Finally, the system is able to match a given smooth trajectory to a known one in the trajectory database and associate its details. Jiang *et al.* [Jia+20] proposed to learn the mapping between a sequence of configuration of two animated characters (*e.g.* inter-character distance, relative orientations) and the sequence of camera viewpoints on these characters. This mapping is referred as *camera behaviors*. In the learning step, first the cinematic features are extracted from multiple RGB movie clips. An LCR-net network [RWS17] estimate the characters poses which is used for (i) extract the camera parameters in the Toric space [LC15] and (ii) extract the relation features between the two characters. These features are given to a gating network that maps the sequence of features to the sequence of camera. Then MoE are able to identify camera motion type by themselves (*i.e.* orbital, traveling, etc...). Based on this work, Jiang *et al.* [Jia+21] proposed a camera motion in-between generator to track 2 characters. As the previous work, the goal here, is to learn from film examples, to track two characters. A network has been design to learn from movie clips, how the camera can transition between two key frames. The toric space coordinates of the camera are extracted from movie clips and then, the network map the *camera motion style space*. From this space, the motion between two given key frames is reproduced according to the one learned.

With the explosion of Deep Learning techniques during the last decade, new perspectives opened for data driven approaches. In the cinematography field, reinforcement learning have helped a lot for the question of motion planning and vision based target identification. Hanocka *et al.* [Han+22] proposed a reinforcement learning strategy for a flying camera to track multiple targets in a team sport environment. The system is able to learn from a few key-frame examples given by an expert user. First, a low-dimensional representation of the scene dynamics is encoded in two networks: the *spatial component* is a variational auto encoder (VAE) and the *temporal component* is an LSTM that compresses the history of the spatial latent states. Then a reward estimator network is trained using a few simulated expert examples: the expert gives a good point of view at the beginning of the epoch and stops the simulation when the point of view is judged to be too bad. Finally, an MPC is performed by sampling sequences

of actions inside the low-dimensional representation and evaluating the corresponding trajectories using the reward estimator. Rucks *et al.* [RK21] proposed a low-depth reinforcement learning approach for a character chasing camera. The objective is for the camera to (i) avoid occlusion and collision (ii) keep the target in the camera frustum and (iii) maintain a reasonable distance from the target. The neural network (*i.e.* Proximal Policy Optimization) is built with two hidden layers of 128 neuron each. There are 48 inputs to the network: the target position and orientation, results of ray casts launched between the camera and the target bounding box corner which account for the target visibility (*i.e.* there is no occluder between the camera and the target and the target are inside the camera frustum) and finally results of 26 rays casts from around the camera in order to sample the local information of the surrounding environment. The network needs to be trained for each specific environment and outputs an action to perform for the camera (*i.e.* a camera translation and rotation). The high noisy output is then smoothed by using a simple interpolation mechanism.

Being able to detect the target on the image space is a critical task for drone cinematography applications. Huang *et al.* [Hua+19b] proposed an expert-based learning framework to generate drone poses in a character tracking task. In the first step three features are extracted from the on-board drone camera: (i) the subject skeleton pose, estimated with OpenPose [Cao+17]; this pose is encoded as thirteen Gaussian masks centered on the skeleton joints; (ii) the background image, which is encoded using a CNN; and (iii) the dense optical flow that encodes the camera motion. In the second step, using these extracted features, a prediction network trained on professional human motion videos infers the next subject pose and dense optical flow. During the last step, from the predicted features, the camera pose is computed, and a feasible path is generated using a min-snap polynomial trajectory algorithm to drive the camera towards the computed pose. Huang *et al.* [Hua+19a] further extended the previous approach by considering a system where the user can input a desired camera pose. Training data are collected from video sequences where the features (*i.e.* 20-frames pose sequences associated to the next pose) are extracted using OpenPose and fed to the learning network. The network learns to predict the next pose from the past sequence using a neural machine translation (NMT). At run-time the system extracts the features from the video captured by the drone. These features are processed by the decoder of the NMT that returns the next pose. Finally, the system generates a command sequence to drive the drone towards the predicted pose. Gschwindt *et al.* [Gsc+19] proposed a reinforce-

ment learning approach for drone cinematography based on a previous work [Bon+18] (see section 2.4.1). A deep Q-network (DQN) is trained to predict the next action to perform for the flying camera. The network is trained using three elements: a height map that represents the environment around the target, the current camera shot type (front/rear/left/right) and the duration within this shot counted in steps, where step has a duration of 6 seconds. The reward function relies on the optimization function used in [Bon+18] that accounts for camera angle, occlusion, collision and on-screen size of target, and is extended to handle shot duration. The network is then able to select an action to perform for the next step: either maintain the current shot type or change to a new shot type. Finally, the trajectory is generated by CHOMP [Rat+09], a motion planner based on gradient optimization, between the current camera location and the location corresponding to the selected action. We refer the reader to [Bon+20a] which summarizes all their contributions [Bon+18]; [Bon+19]; [Gsc+19] and integrates real-time generation of the height map using LiDAR.

Loquercio *et al.* [Loq+21] proposed a learning-based path planner for high-velocity quadrotor travelling in cluttered environments. The process works through two stages: an offline training stage, and an online optimal control stage. The offline training stage is fully performed in simulation. A privileged expert is devised, which has full knowledge of the environment (*i.e.* represented as a point cloud) and of the state of the drone (*e.g.* orientation, altitude and a depth map computed from its point of view) along a reference collision-free trajectory between two positions [Liu+18]. From these inputs, the expert randomly generates thousands of collision-free trajectories for the next second, using a Metropolis-Hasting algorithm [FB13], then outputs the best three trajectories regarding their collision probability and the distance to the reference trajectory. Then, a student network learns how to imitate the privileged expert output, with only the drone state as input. In the online stage, at each time step, all three trajectories output by the trained network are represented as B-spline curves, and evaluated according to their collision probability learned from the expert network. The one with the lowest cost is finally tracked using MPC.

2.4.5 Trajectory evaluation

During the last decade and a half, the two research fields of virtual cinematography and drones have converged. Before 2008 most of the works in virtual cinematography

focused on camera positioning (*i.e.* viewpoint) [CON08]. At first most of the works focused on the amount the information given by viewpoints or sequences based on the notion of viewpoint entropy as defined by Vazquez [Váz+01]. With the evolution of graphics performances, 3D environments became denser and more dynamic. From this point the community of virtual cinematography started to focus on path planning problems and how to generate aesthetic trajectories. On the other hand, the topic of drone cinematography has emerged as a promising research field. Drone (or *aerial*) cinematography has stemmed from robotics where most of the contributions focused on feasibility and trajectories optimization for path planning. Almost all contributions we show in this state-of-the-art report has a way or another to judge either camera view point or the camera trajectory generated by their system. In this section we overview how the trajectories are evaluated and what are the critical properties to ensure to generate "good" trajectories. We will decompose this analysis in two axes, first we analyze the optimization process that most of the works rely on, then we analyze the different user evaluation techniques proposed in the literature.

Optimization function analysis

Automatic cinematography is, for the most part, an optimization problem, and we can define the optimization function as a combination of quantifiable properties. Most of the time we search to minimize the energy of these combined properties 2.1.

We can break down these properties in 4 categories: (i) the environment related properties, (ii) the camera motion properties, (iii) the on-screen properties and (iv) the on-screen motion properties.

Environment related properties: These properties are what the camera will have to be aware of when moving around in the scene. One of the most studied properties in automatic cinematography is the visibility and occlusion avoidance. Contributions such as [Osk+09]; [BY10b] proposed a global planning approach to compute visibility for virtual camera control on static scenes. Ranon *et al.* [RU14] proposed an accurate visibility computation between a camera and a target. This works also shows that ray cast is more accurate than image-based techniques for visibility computation. Due to the highly constrained nature of camera control, visibility can become computationally heavy if computed for dynamic environments and over time while rendering is less expensive. Christie *et al.* [CNO12] proposed a rendering based occlusion avoidance technique for

multiple targets. When referring to environment related properties for camera planning, and especially for drone cinematography, collision is a critical point. In the drone cinematography field [Geb+16] proposed to optimize a set of user defined key frames to compute a feasible and collision free trajectory for a quadrotor. Obstacles are approximated as spheres and added to the optimization constraints. Later, [Bon+18] used a LiDAR sensor to voxelize the environment and launch rays between the camera and the target anticipated motion to avoid collision and occlusion. And more recently [Loq+21] used data-driven approach to learn to avoid collision for a high velocity motion drone.

Camera motion properties: The smoothness of a camera motion is a key property that was addressed in both fields. In virtual cinematography, camera motions are usually physically plausible while in drone cinematography, feasibility constraints need to be considered. On the one hand, optimizing the trajectory to minimize the third motion derivative (*i.e.* the jerkiness) has become common in order to produce smooth trajectories. In this idea [Gal+13]used force based steering system to move the camera toward an ideal position. In another way [Osk+09] used Hermite curves to ensure a smooth camera trajectory. Gebhardt *et al.* [Geb+16] proposed an optimization process to generate smooth and feasible trajectories for quadrotors by minimizing the energy function of the motion model. Driven by the same idea yet using MPC optimization [Bon+18]; [Näg+17a] proposed a system to generate in real-time collision-free trajectories for drones. On the other hand, [Kur+14] consider the camera shakiness as an element of style and try to learn and apply some degree of shakiness to camera trajectories (*i.e.* handheld camera for example).

On screen properties: Maintaining on-screen properties as image composition or view angle, has often been considered in the optimization process. Contributions such as [Abd+11]; [LT17] encoded numerous standard cinematographic rules (*e.g.* framing, shot type) on their system. Another way to maintain these properties is to use a derivable function defined by the on-screen properties to describe the camera position in the 3d environment as the potential fields. Yeh *et al.* [Yeh+11a] defined multiple potential fields from on-screen properties such as view distance or view angle. Lino *et al.* [LC15] introduce the toric space, an intermediate camera space built to solve the framing for one or a pair of targets. Relying on the toric space [Gal+15a] proposed an optimization based method to build camera rails to maintain image composition. Moreover, [Jia+20]

also rely on the toric space but with a data driven approach. After extracting some movie clip feature, they train a neural network to reproduce similar camera behavior as movies inside virtual environment. Nageli *et al.* [Näg+17c] proposed an automatic drone system that optimizes the trajectory according to cinematography guidelines like targets view angle and shot type.

On screen motion properties: Finally, since cinematography is the art of **motion picture**, it is also important to account for the on-screen motion. Unfortunately, these properties remain under-addressed in virtual cinematography and drone cinematography. Among the few works addressing this aspect, Argelaguet *et al.* [AA10] used optical flow in addition of image saliency that provides data on the image luminance and chrominance variation. Huang *et al.* [Hua+16] proposed to use the optical flow to adapt the camera motion speed to devote less time to the less interesting trajectory parts.

In image analysis field, there are other properties that can be accounted for and interesting to use, such as image saliency which has proven to be really important for film understanding. As Bruckert thesis [BCM21] shows, visual saliency in films is important to understand what are the viewers focuses while watching a cinematographic production. In virtual cinematography, these kinds of properties haven't been investigated much.

User study analysis

While there are some quantifiable properties that can be computationally optimized, cinematography styles and quality is a subjective matter. Thus, performing user studies provides a means to assess the visual quality of camera motions. In practice only a few contributions in virtual and drone cinematography fields have performed extensive user studies. First user studies were proposed by [AWC10]; [Yeh+11a] where 30 computer graphics students have been asked, given a pool of videos, which one they find the most appealing. In Yeh *et al.* [Yeh+11b] from 12 overview sequences rendered using their system, a baseline method or a professional sequence, 70 users were asked to choose their preferred videos. This was performed for 12 pairs of videos to reduce as much as possible the bias. The ranking relied on 3 criteria: (i) the more stable, (ii) the more professional and (iii) the more narrative content. In [Bon+20a] 10 participants were asked to rank different videos of 30s captured with different policies from most to least visually appealing. In [BBS20] 15 participants were asked to choose which sequence they

prefer. Each sequence is rendered to have a change in view angle every 3s, among a predefined set of angles (*i.e.* 45°, 90°, 135°, or 180°). Other means of evaluation rely on cinematography experts to validate sequences generated by the systems. In [Näg+17b] an expert was asked to give feedback on their system to validate the interface and the trajectory design system. Galvane *et al.* [Gal+18a] did both user study and expert validation; 12 participants were asked to compare their controller to a baseline controller for 6 criteria: familiarity, ease to use, fluidity, framing, precision, and satisfaction. The expert user where asked to use the system for 30 minutes and give feedback. Finally, in [Bon+20b] the authors tried to create a latent space of the camera trajectory emotion. They generated multiple flying camera shots with different view angles and asked 15 users to rank 70 captured clips along emotional axis.

2.5 Conclusion

In this chapter we presented an overview of the computational virtual cinematography field. We identified gaps in the literature that we will try to cover in this thesis and the future challenges for the community.

One can view the automated approaches as composed of three approaches:

— The **local** approaches are computationally efficient but easily trapped in local minima due to the local knowledge they have. The navigable space is reduced to the local surrounding of the camera the goal being to find the immediate next camera configuration.

— The **global** approaches need heavy pre-computations to process and store the environment data in dedicated data structures. A set of camera configuration will then be optimized using the global information.

— The **hybrid** approaches are a mix of both previous approaches and there are no precise guidelines to build hybrid approaches. The community however, tends to agree on the use of the MPC technique that is able to find a valid solution for a defined time horizon then recompute at a higher frequency in order to account for local changes.

Now that we have analyzed the field of virtual cinematography, we can see that there is currently no solution that satisfies our constraints at the same time (**Automatic camera target(s) tracking** for **real-time application** accounting for **dynamic** changes

of the virtual environment). Global approaches have a need of heavy pre-computation and because we need to account for dynamic elements we would have to update the pre-computed data structures which is too expensive for us. Local approaches could be a good way to satisfy our constraints. Efficient solutions that handle well visual properties (*e.g.* the Toric Space [LC15]) do not handle or solve occlusion avoidance.

In this thesis, a first problem we address is how to handle simultaneously occlusion and visual properties (*e.g.* distance, angle, framing) in a target tracking task and in real-time. Then, in the second problem, we address the limitations of the first proposition. We rely on an MPC-based approach coupled with a dedicated parametric space of camera motion to characterize the trajectory quality.

CHAPTER 3

A LOCAL APPROACH FOR OCCLUSION-FREE VIRTUAL CAMERA IN THE TORIC SPACE

As shown in the previous chapter, in most interactive applications, camera are traditionally controlled with low level interaction metaphors which offer limited degrees of control on the camera and controlling these cameras is still a challenging task for non-expert users. Furthermore, solutions remain limited in handling occlusion and collision avoidance in the literature. Thus, there is a need to provide a higher level of camera automation Therefore, in this chapter we address these 4 challenges: (i) the camera has to be fully automated; (ii) the system has to be efficient enough to run in real-time applications; (iii) the camera has to track target(s); (iv) the system needs account for dynamic elements in the scene.

Therefore, In this chapter we address these challenges by proposing a real-time cinematic camera control system for fully dynamic 3D scenes, which maximizes visibility of one or two targets with no pre-computation. To do so, we leverage the contributions of the Toric space (*i.e.* [LC12]; [LC15]) which offer an efficient way to address cinematic camera placement, yet without addressing the issues of visibility computation/maximization nor quality of camera motions. The contributions are:

— an efficient shadow-mapping algorithm to compute visibility information into a Toric space coordinate system;

— a customizable anisotropic blurring algorithm to generate, in Toric space, an *occlusion anticipation map*;

— a texture-based style model to encode viewpoint preferences (*i.e.* directorial styles in viewpoints can be expressed as *textures*);

— a physically-plausible camera control model enabling, from this information, to create smooth camera motions in real-time. We supplement this model with a

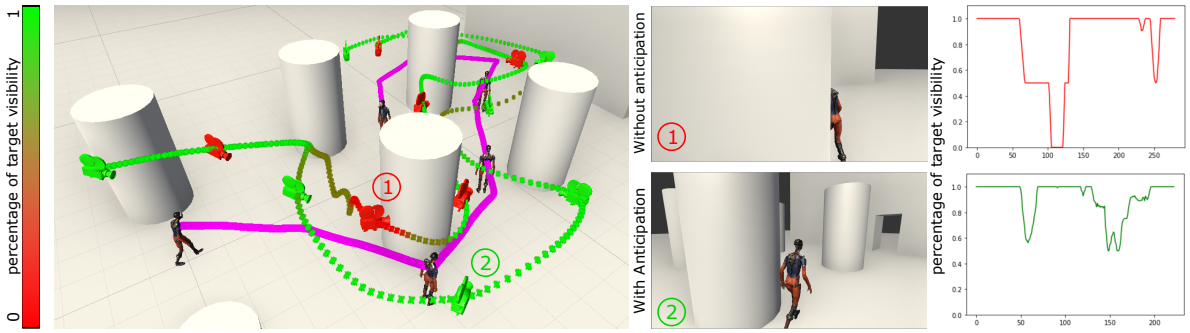


Figure 3.1 – Real-time camera control system automatically anticipates and avoids static and dynamic occluders. Starting from the same frame, our system without anticipation activated (red camera) is unable to anticipate, hence remains occluded until the character moves away from the column (middle image 1); using our anticipation technique (green camera), our system enables the camera to predict the occlusion, and move to an unoccluded view of the character (middle image 2).

set of strategies (*e.g.* introduce cuts) to handle symptomatic cases (*e.g.* when no occlusion free viewpoint can be found locally).

3.1 Background

Positioning cameras in 3D scenes is highly connected to the notion of viewpoint entropy [Váz+01], *i.e.* measuring the quality of a viewpoint with regard to the amount of information it conveys for a given scene. When addressing the problem of moving cameras that should follow moving targets in a cinematographic way, three criteria are critical: (i) maintain enough visibility on these targets, (ii) provide viewpoints in which the visual arrangement of targets follow common aesthetic rules used by cinematographers [TB09b] and (iii) provide smooth camera motions because beside the notion of viewpoint quality, the notion of motion quality is also an important point to address.

3.1.1 Toric space

Most approaches have tackled the framing problem as a search or a constraint-solving problem in a 7D camera space (*i.e.* optimizing a 3D position, 3D orientation, and field of view for each camera, and each time step). One key issue is the strongly non-linear relation between the low-level camera parameters and the visual arrangement constraints (*e.g.* targets on-screen positions, size, or viewing angles) to satisfy.

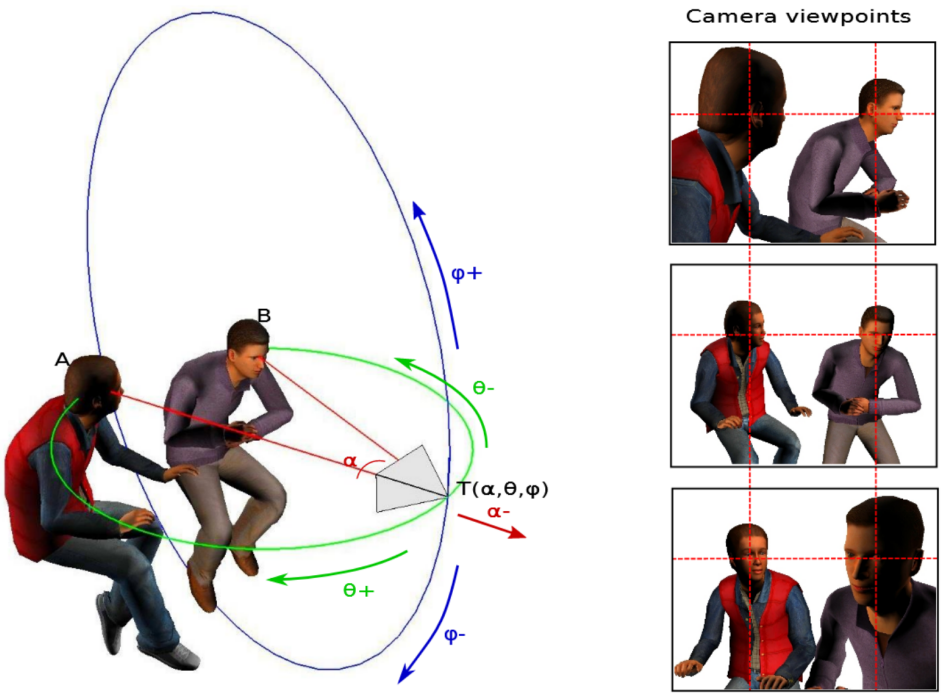


Figure 3.2 – Toric space representation proposed by [LC12]; [LC15] to algebraically solve combinations of on-screen constraints. A viewpoint is parameterized with a triplet of Euler angles (α, θ, φ) defined around a pair of targets (A and B); α (red) defines the angle between the camera and both targets – it generates a spindle torus on which to position the camera –, θ (green) defines the horizontal angle and φ (blue) the vertical angle around the targets.

Hence, for efficiency, it is often required to reduce the dimension of the problem, by either optimizing the camera position and orientation in two separate steps or by constraining the possible camera positions. The Toric space proposed by Lino and Christie [LC12]; [LC15], is a 3D compact space in which some visual constraints can be directly encoded (in terms of both camera position and orientation) or solved algebraically. This space is defined as a continuous set of spindle torus, around two targets. A 7D camera configuration is fully determined by 3 parameters: an angle (α) algebraically computed from the desired targets on-screen positions – any camera position (and its associated orientation) on this α-Toric surface satisfies this key constraint –, as well as a horizontal and a vertical angle (θ and φ) on the torus (see figure 3.2). Furthermore, visual constraints can be robustly and interactively enforced or manipulated, while this is a difficult problem in the 7D camera space. Yet, visibility information remains not straightforward to compute in Toric space, which is a key motivation of this work.

3.1.2 Visibility for computer graphics

Visibility (and by extension shadow) computation in 3D scenes has received much attention in computer graphics, mainly for rendering purposes. An extensive overview of general visibility and shadow computation is beyond the scope of this work, though a good insight in visibility problems classification can be found in [Coh+03], and overviews of modern shadow computation techniques can be found in these two works [WP12]; [Eis+16]. Hence, we here only review techniques that are relevant to our domain of interest.

Ray-casting techniques Rely on direct ray-primitive intersections [Rot82] to check if some geometry exist between two 3D points. This is mostly used in path-tracing techniques, as it allows computing point-point visibility very efficiently. By casting many rays, one can obtain a rough approximation of the from-point visibility of a complex geometry. It however remains more expensive than rasterization, as it requires to cast many rays, and to dynamically maintain a bounding-volume hierarchy. One reason is that it still heavily relies on CPU computations (at least until hardware ray-tracing GPUs become the norm).

Rasterization-based techniques Rely on the hardware graphics pipeline, which offers very powerful tools (*e.g.* the depth map or z-buffer). It enables approximating *shadow maps* [Wil78] by computing a depth map from a light source, or even *soft shadows* [HH97] by using multiple shadow maps computed from sampled points on an area light source. [Rit+08] reversed this concept by computing many low-cost (128x128 pixel) shadow maps from sparse 3D scene points. They approximate the scene as a set of sample points, split them into subsets, and render each subset into a shadow map, with a point-based rendering method. As a result, they obtain inaccurate shadow maps. These maps provide a very poor approximation of direct visibility, but are good enough to efficiently approximate indirect visibility (illumination and shadows) from punctual or area light sources. Our technique is inspired by the shadow mapping concepts, which remain cheap (*i.e.* the z-buffer is encoded in the hardware of most GPUs) and provide a precise-enough approximation of the from-point visibility in complex and dynamic scenes. More generally, we design our computation pipeline to be massively parallel (*i.e.* shader-oriented).

3.1.3 Visibility for camera control

Visibility is also a central issue when controlling virtual cameras, though its has been paradoxically under-addressed in this domain. Different concerns must be reconciled: (i) the level of precision (from rough to precise approximation), (ii) whether dynamic occluders or complex geometries are accounted for, and (iii) whether the camera can anticipate on future occlusions. In most existing works, the visibility is computed by casting rays toward a rough geometric abstraction (often a bounding box or sphere) of targets, with no or few anticipations. We hereafter focus on techniques handling visibility for camera motions.

Local visibility planning To plan paths in scenes with dynamic occluders, researchers have proposed reactive planning techniques. Inspired by occlusion culling methods used in rendering, Halper *et al.* [HHS01] proposed an occlusion avoidance relying on *Potential Visibility Regions* (PVR). They firstly define preferred viewpoints as a set of bounding spheres. Each sphere is shaded regarding its preference level (the higher, the brighter the color). They secondly use a depth buffer from the target position. They render all volumes from most to least desirable, while considering the depth of occluders. As output, they generate an image buffer whereby the brightest color (*i.e.* most desirable camera position) that first pass the depth test is visible. To anticipate occlusions, they also consider the past occluders trajectories and accelerations, and solve the camera for that predicted state. In turn, the camera path is adapted so that the camera can be at this predicted position at the prediction time. Christie *et al.* [CNO12] extend this concept to two targets. They first compute low-resolution depth buffers from a large sample of points on each target, in the direction of the camera. They combine these depth buffers to create visibility volumes around the camera. Furthermore, they also extend it to three or more targets [CON08] by combining pair-wise computations, and aggregate visibility in a temporal window to avoid over-reactive camera behaviors. Our technique relies on similar concepts, while we project results in a configuration space (the Toric space) allowing to also solve for the viewpoint semantics. Local approaches are best suited for planning paths step-by-step. Hence, if no solution exists in the rendering, they will fail to find an occlusion free position (even if one exists). In turn, they cannot plan a global path to this position. In our approach, we overcome this problem by considering more global visibility information, to make cuts, as an alternative when no local solution is found.

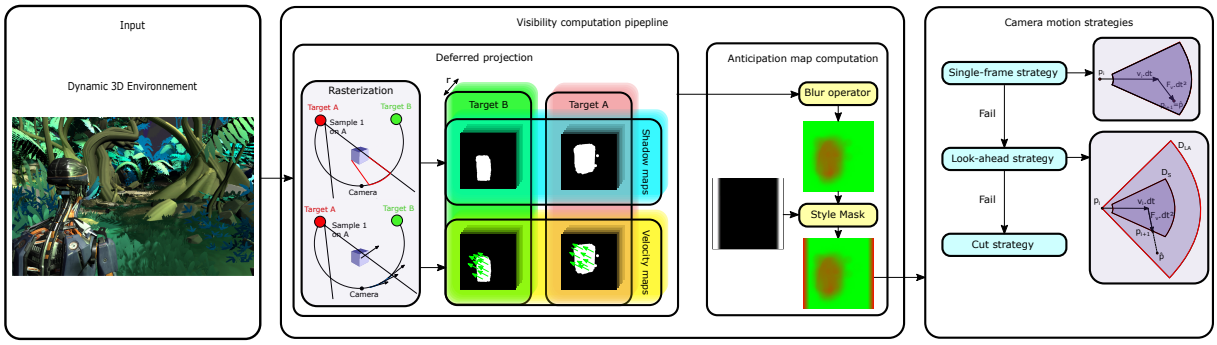


Figure 3.3 – Overview of the system: (left) a dynamic 3D scene serves as input, (middle) our visibility anticipation process consists in computing shadow maps together with occluder velocity maps for each target, and combining them to construct an occlusion anticipation map (right) we use this anticipation map together with a physical camera model to both maximize visibility and camera motion smoothness.

Visibility in Toric Space Remains an under-addressed problem. [LC15] proposed to compute a set of camera positions in Toric space, satisfying other visual constraints. They convert them back to Cartesian space, evaluate targets visibility with ray-casting, and discard those with low visibility. The main drawback is that their visibility computation remains imprecise and costly, while they target placing cameras in static scenes only. To our knowledge, there has been no other attempt to compute visibility, and none to anticipate occlusions in Toric space.

3.2 Overview

The core of our method consists in solving an optimization problem in a 6D camera space (position and orientation) in real-time that accounts for 3 main criteria: (i) computing a sequence of viewpoints that satisfy a user-specified visual arrangement of targets (*e.g.* desired on-screen positions), while (ii) maintaining as much visibility on the filmed targets as possible, and (iii) ensuring that camera motions remain as smooth as possible. First, by using the Toric space coordinate system [LC15], given two desired on-screen positions for two distinct targets, the camera orientation is fully determined by a given position in the Toric space. We leverage this property to re-write the overall problem as a constrained-optimization on the camera 3D position only (rather than a 6D).

The Toric space being defined by two targets (abstracted as two 3D points), we

propose a tracking with only one target but rely on the classical gaze-room filming convention [TB09b] by considering another 3D point placed in the forward direction of the target that represents the target's anticipated 3D location. By composing on the screen, the target together with its anticipated position, we provide enough visual space ahead of the character, hence satisfying the gaze-room convention. The desired on-screen positions of these two targets is enforced by maintaining the camera on a given Toric surface within the Toric space [LC12]. This addresses criterion (i).

To address criterion (ii) (visibility), a key requirement is the ability to efficiently evaluate the visibility of our two targets for many camera configurations. To further maximize this visibility over time, we need to anticipate the motions of the camera, targets and occluders, *i.e.* computing the visibility ahead of time.

In order to do so, we propose a 2-stage computation pipeline (see figure 3.3). In the first stage (deferred projection), we draw inspiration from the shadow mapping technique to compute the visibility of both target objects. We render occluders from the viewpoint of each target and project this information onto the current Toric surface (see figure 3.2). We therefore obtain a 2D scalar field on this surface which we refer to as a **target shadow map**(or **S-map**). The field encodes the composed visibility of target objects at the current frame. Then, in addition, we compute the velocity of our occluders and generate a corresponding 2D vector field in the Toric space which we refer to as the **occluder velocity map**(or **V-map**). In the second stage, we combine the **S-map**and the **V-map**to compute a new 2D scalar field corresponding to an anisotropic blur of occlusions along the occluders' velocity directions given by **V-map**. In a word, we compute the predicted visibility of the occluders knowing their velocity into a scalar field to which we refer as **occlusion anticipation map**(**occlusion anticipation map**or **A-map**). This encodes the probability of future occlusion of the targets, expressed in Toric space, within a given time window.

To address criterion (iii) (smooth camera motions) we propose the design of a physically-driven camera motion controller in which our camera is guided by a number of external forces. We start by defining a search area which approximates the locus of future camera positions, given a time window and our physical camera model. We then sample points in the search area and project them on the Toric surface. For each projected point, we extract the anticipation information stored in **occlusion anticipation map**, and then decide where to move the camera. To handle specific cases, different camera motion strategies have been defined (see Section 3.4). This addresses the

Table 3.1 – Notations

$(\mathbf{u}, \mathbf{v})$	angle between two vectors
$\mathbf{u} \cdot \mathbf{v}$	dot product
o	target object
s	sample point index
r	number of sampled points on target object o (*i.e.* $s \in [1 \,..\, r]$)
i	camera motion iteration
(x, y)	pixel/texture coordinates $\forall x, y \in [0, 1]$
$T(x, y)$	value in a 2D buffer/map T
S_s^o	map computed for sample point s on object o

combination of criteria (i) and (iii).

3.3 Visibility computation pipeline

We detail the two stages of our visibility computation: (i) computing pairs of **target shadow map**and **occluder velocity map**for the two targets; (ii) combining both maps to generate an **occlusion anticipation map**. Both stages rely on a mapping between a Toric surface and a texture.

The texture is mapped using the toric parameters θ (horizontal angle) and φ (vertical angle). Before performing the projection, we build a mesh representation of this surface on which every vertex is supplied with two additional channels of information: (a) its texture coordinates, that will linearly map the Toric angles (θ, φ) illustrated in figure 3.4, (b) the normal and tangent to the surface at this vertex that compose the local surface basis matrix TBN which is later required in the camera motion strategy.

We rely on these additional channels to (a) transform any local pixel of a projected map into its corresponding Toric texture coordinates (as illustrated in figure 3.5), and (b) transform an occluder velocity expressed in the Cartesian space, into a plane tangent to the Toric surface (*i.e.* a local Toric space velocity vector). With this in mind, we can detail our visibility computation pipeline.

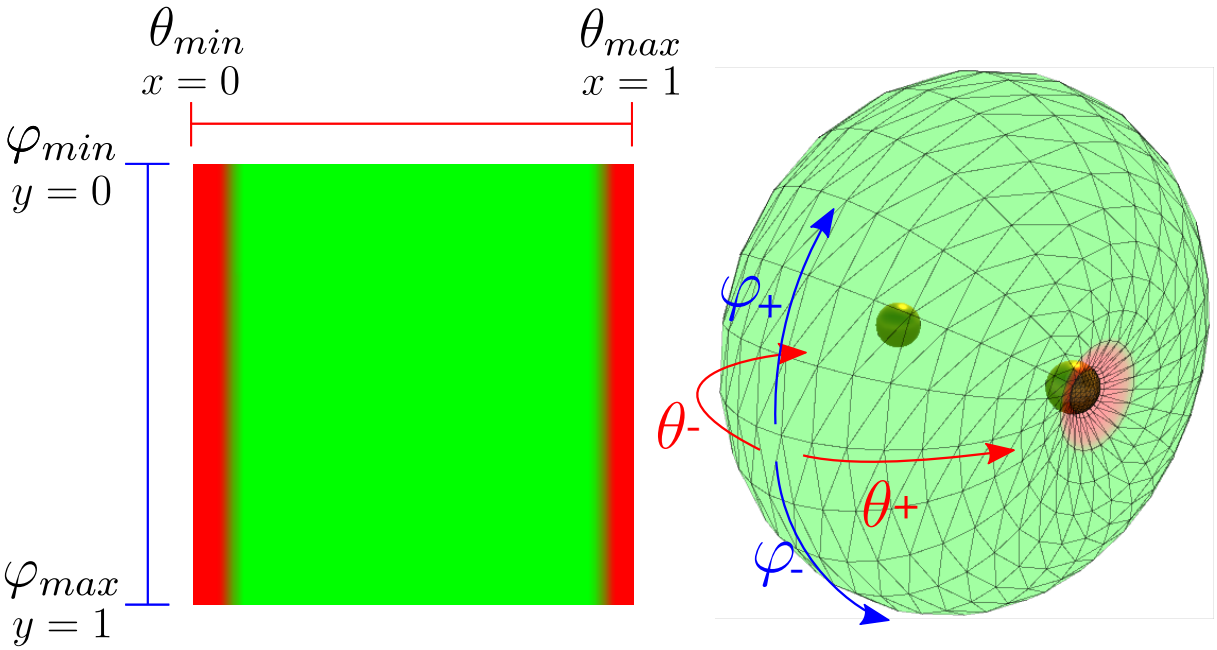


Figure 3.4 – (x, y) texture mapping expressed in the Toric space (θ, φ).

3.3.1 Deferred projection of shadows and velocities

In order to obtain a good approximation of the visibility of potentially complex 3D targets, we propose to perform visibility computation for a number (r) of random sample points on the surface of each target object (inspired by [CNO12]). Let's consider a sample s, picked on target object o. For this point, we will render the **S-map** (S_s^o) as well as the **V-map** (V_s^o) using deferred projections.

G-Buffers To compute these two projections, we first need to render required information into a set of screen-space textures. We perform two render passes, both using the same camera projection (looking from s to the current camera position on the Toric surface): the first render pass only renders the mesh of the Toric surface, and the second renders the occluders (*i.e.* the rest of the scene without the object o to avoid self-occlusions). In the first pass, we store the depth map in texture Z_{toric}, the mesh 3D positions in P_{toric}, and the surface texture coordinates provided in the extra information channels on the mesh in texture UV together with its normal and tangent vectors (in two textures N and T). The P_{toric} map is later used to compute the velocity of the Toric by finite differences at the current location. In the second render pass, we store the depth map (in texture Z_{occ}) and the occluders 3D positions (in texture P_{occ}). The infor-

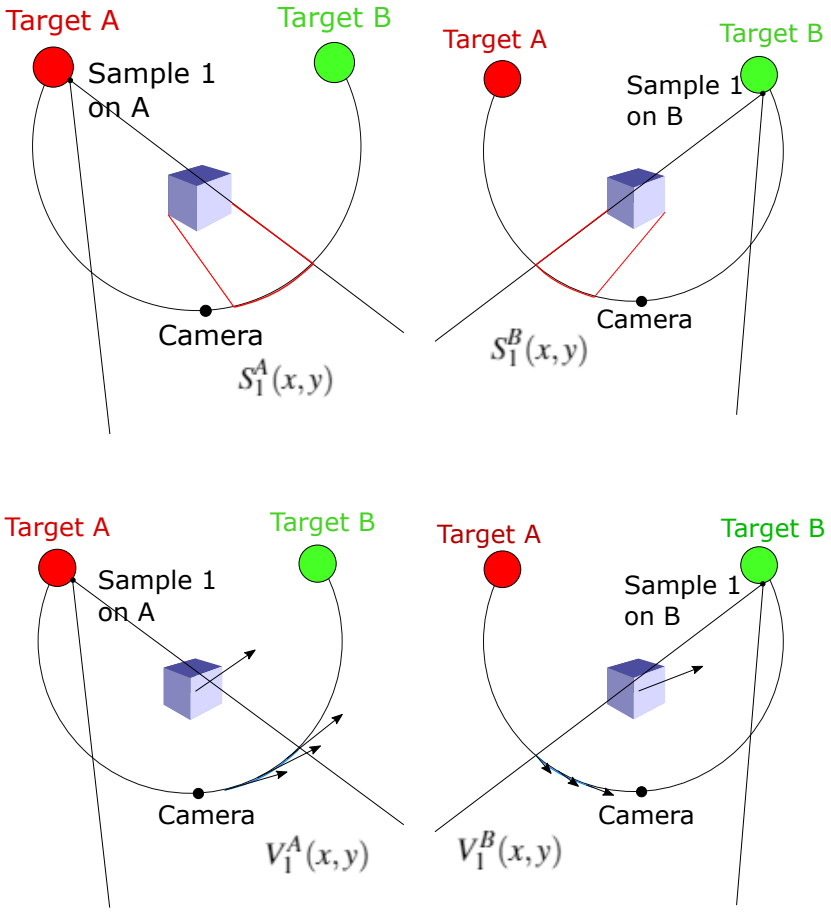


Figure 3.5 – Deferred projection performed from one sample on each target, with at the top the shadow maps, and at the bottom the velocity maps

.

mation stored in the map $UV(x,y)$ is required to express a local position (x,y) of a map into the corresponding global position in Toric coordinates on the **S-map**and **V-map**.

Target shadow maps

For each point s on the target object o, we fill the 2D **target shadow map**by simply comparing depths:

$$S_s^o(UV(x,y)) = \begin{cases} 1 & \text{if } Z_{occ}(x,y) < Z_{toric}(x,y) \\ 0 & \text{otherwise} \end{cases}$$

Occluder velocity maps

We now want to predict locations from which this sample point will be occluded in the next frames. To do so, we will rely on the current occluders velocity. What we propose is to first render this velocity into a velocity map (V_s^o). By deriving our position buffer P_{occ} using finite difference between two frames, we determine the occluders world-space velocity as $\mathbf{v}_{occ}(x,y) = dP_{occ}(x,y)/dt$. In the same way, we determine the world-space toric velocity as $v_{toric}(x,y)$. We finally express the occluders velocity in the Toric space:

$$V_s^o(UV(x,y)) = TBN^t.(\mathbf{v}_{occ}(x,y) - \mathbf{v}_{toric}(x,y))$$

Where the matrix TBN is computed from the normal and tangent vectors $N(x,y)$ and $T(x,y)$. Do note that by removing the torus velocity, we get the relative velocity with respect to the torus (moving with targets). Then, by projecting it in the tangent space, we can in fact remove its 3rd component (orthogonal with the normal) to obtain a 2D vector expressed in its tangent plane, which we store in the map.

3.3.2 Occlusion Anticipation Map computation

We now have generated $2r$ pairs of shadow+velocity maps (one for every sample point). We combine all maps to generate a single **occlusion anticipation map** $A(\theta,\varphi)$ encoding the area of probabilities of our target objects being occluded or not (from 0 – none will be occluded – to 1 – both will always be occluded). Do note that $shadow + velocity$ maps contain information only for a region of the torus, typically where the information has been projected around the current camera location. So at this stage, we will only update the **A-map** in these regions.

Now, in order to account for uncertainty in the future location of occluders, we propose an *occlusion prediction model* which relies on classical image processing operators for efficiency.

Given a point s, sampled on a target o, we can easily check whether it is (or not) occluded from a location (θ,φ) by reading the associated value $S_s^o(\theta,\varphi)$. Knowing the occluder's velocity (*i.e.* $V_s^o(\theta,\varphi)$), we propose to estimate the amount of future occlusion through an uncertainty function U, which in practice is an anisotropic blur operator, directed along the occluder's velocity. The operator takes as input a pair of 2D vectors

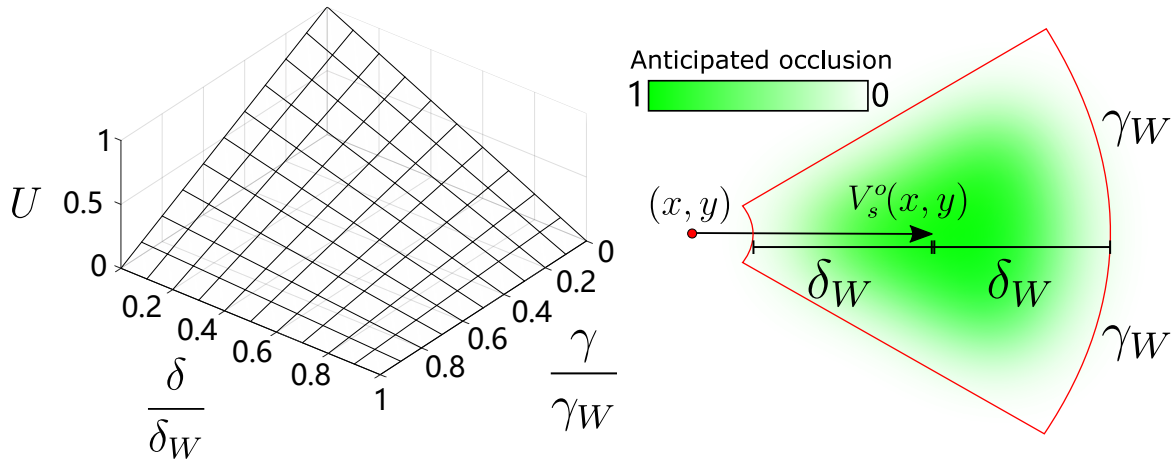


Figure 3.6 – Uncertainty function U. (left) graphical representation of U accounting for the distance δ and angle γ to the velocity vector V. (right) blur kernel in texture space for a pixel at position (x, y); the length and angular cutoffs are marked as red lines.

(expressed in Toric space):

$$U(\mathbf{v}_1, \mathbf{v}_2) = U_l(\delta).U_a(\gamma) \tag{3.1}$$

where $\delta = \|\mathbf{v}_1\| - \|\mathbf{v}_2\|$ and $\gamma = (\mathbf{v}_1, \mathbf{v}_2)$.

In other words, for a neighbor location (θ', φ'), we compare the length and angular differences of the extracted velocity $\mathbf{v}_1 = V_s^o(\theta, \varphi)$ and vector $\mathbf{v}_2 = (\theta', \varphi') - (\theta, \varphi)$. In practice, we cast both comparisons into falloff functions (as illustrated in figure 3.6):

$$U_l(\delta) = max\left(1 - \frac{|\delta|}{\delta_W}, 0\right), U_a(\gamma) = max\left(1 - \frac{|\gamma|}{\gamma_W}, 0\right)$$

where δ_W and γ_W are custom parameters that represent the length and angular cutoff values of our uncertainty function (in our tests, we used values given in table 3.2).

Applying the prediction model Given our prediction model for one point s on one object o, we can now focus on computing the probability of occlusion from any possible camera location, *i.e.* $A_s^o(\theta, \varphi)$. In theory, this can be computed by integrating function U over the couple S_s^o and V_s^o:

$$\iint_{-\infty}^{+\infty} U\Big(V_s^o(\theta + x, \varphi + y), (-x, -y)\Big) S_s^o(\theta + x, \varphi + y)\ dx\ dy$$

But in practice, we rely on textures. Hence, we compute its value at a pixel-wise level, through a double sum over pixels of S_s^o.

Aggregating predictions for all samples Our $2r$ partial occlusion predictions (each for a single point s) can now be aggregated into an overall **A-map**:

$$A(\theta, \varphi) = \frac{1}{2r} \sum_{o \in \{A,B\}} \left[\sum_{s=1}^{r} A_s^o(\theta, \varphi) \right]$$

3.4 Camera motion

We have all required information to design a camera motion controller that can enforce on-screen constraints (this is the purpose of our supporting Toric surface) and avoid future occlusions of target objects. Put altogether, we provide a low-dimensional camera space (the 2D Toric surface), supplied with a scalar field encoding the risk of our targets being occluded in the next frames.

In the following, we propose to mimic a physical model [Rey99]; [Gal+13] where our camera will behave as a particle on which external forces are applied steering it toward an occlusion free locations. The motion of the camera can therefore be formulated as a function of its previous position ($\mathbf{p_i}$), velocity ($\mathbf{v_i}$) and k external forces ($\mathbf{F}$):

$$\mathbf{v}_{i+1} = \mathbf{v_i} + \left[\frac{1}{m} \sum_{j=1}^{k} \mathbf{F}_{i+1}^j \right] dt \quad \text{then} \quad \mathbf{p}_{i+1} = \mathbf{p_i} + \mathbf{v}_{i+1}.dt$$

with dt the time step, and m the particle mass. Note that we also need to ensure the camera remains on the surface of our moving torus.

Physically plausible camera motion To enforce visibility at a low computational cost, we propose an ad-hoc position update, which is nonetheless explained by a physical model with two external forces:

— a visibility force ($\mathbf{F}_v$) that steers the camera towards a location with a lower occlusion risk;

— and a damping force ($-c\mathbf{v_i}$) controlling the camera's reactivity (the higher the damping factor c, the lower the reactivity).

We firstly observe that, in a physical model

$$p_{i+1} = p_i + \left[v_i \left(1 - \frac{c}{m} dt \right) dt \right] + \frac{1}{m} \mathbf{F}_v . dt^2 \tag{3.2}$$

the middle part is a fixed translation, function of the velocity, damping, and elapsed time. The right part represents a bounded surrounding area, function of the two external forces and the elapsed time, containing all possible camera locations at the next frame (or if we increase dt, after a few frames). Starting from this observation, our general strategy is to search within this surrounding area for a less occluded camera location. This should provide a physically-plausible camera motion, near-optimal in the sense of the three constraints we stated earlier (smooth motion, on-screen composition, and occlusion avoidance).

In order to move the camera to a potentially less occluded location (if one exists), we propose three local sampling strategies, applied in this order:

1. we use a *single frame* search D_S for a less occluded camera location in the next frame $(i + 1)$. We therefore restrict the search to the surrounding area which the camera can reach at the next frame (see figure 3.9a).
2. When D_S finds no satisfying solution (*i.e.* a configuration less occluded), we use a *look-ahead* search D_{LA} in a wider surrounding area that the camera can reach after a given fixed number of frames. We then steer the camera, by selecting the closest position inside area D_S (see figure 3.9b).
3. When D_{LA} still finds no satisfying solution, we search a *cut* to a further, less occluded, location. We cast this into a stochastic search in area D_C (in practice the whole surface) and then instantly teleport the camera to this position; this is a common strategy in video games as well as movies (camera "cut" policy [TB09a]).

Note that our **A-map** will here act as a regularization operator in these local searches, by casting high-frequencies contained in the **S-map**s into a low-frequency prediction valid for a few frames ahead.

Search process We evaluate the anticipated occlusion at position $\mathbf{p}(\mathbf{v}_i)$ (*i.e.* if no external force applies), and then randomly sample locations in a given surrounding area D_X. The selected location $\hat{\mathbf{p}}$ must provide an improvement in the predicted occlusion:

$$\hat{\mathbf{p}} = \underset{(\theta,\varphi) \in D_X}{\arg\min} A(\theta, \varphi), \text{ subject to } A\left(\hat{\mathbf{p}}\right) \leq A\left(\mathbf{p}(\mathbf{v}_i)\right)$$

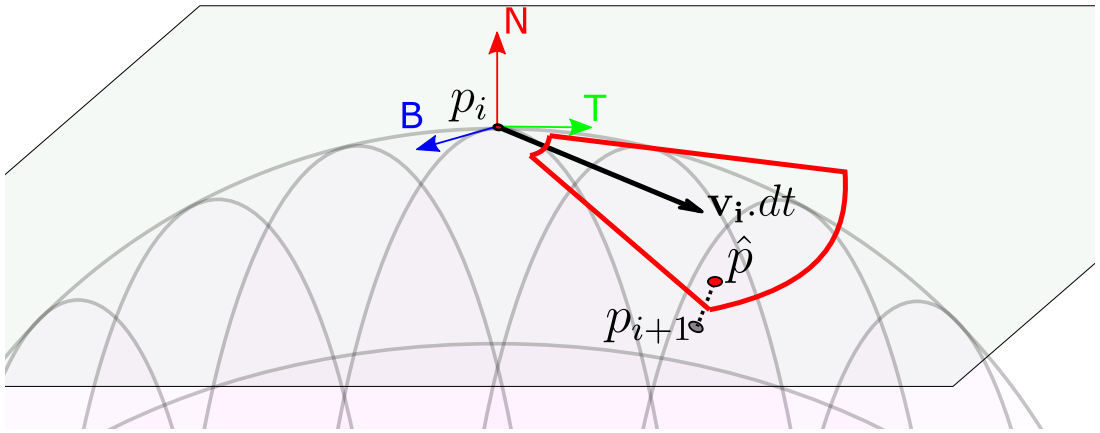
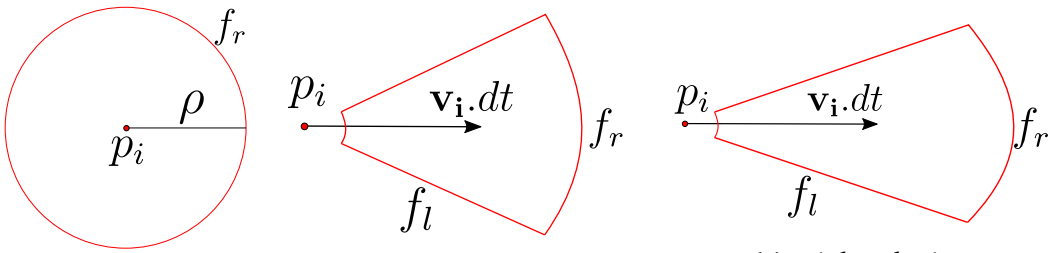


Figure 3.7 – 3D view of our local search process for one frame.

Moving cameras in Toric space A purely physical model of camera motion is defined in world space, while our **A-map** is expressed in Toric space along the coordinates (θ, φ). An easy way to define our camera motion model would be to compute the motion directly in Toric space coordinates. However since the mapping introduces distortions, *i.e.* the same amount of motion in Toric coordinates does not correspond to the same amount of motion in Cartesian space, depending on where the computation is performed on the surface and on the size of the Toric surface. The size is dependent on the Toric α value (angle AsB) and the distance 3D between the targets. We instead propose to define the camera motions in Cartesian world coordinates using a plane tangent to the Toric space. We first define the search area on the tangent plane of the torus at the current camera position; we then uniformly sample points in the search area, which we project onto the Toric surface, *i.e.* to now obtain plausible 2D locations in Toric coordinates (θ,φ) so to exploit the information held in the **occlusion anticipation map**. This process is illustrated in figure 3.7.

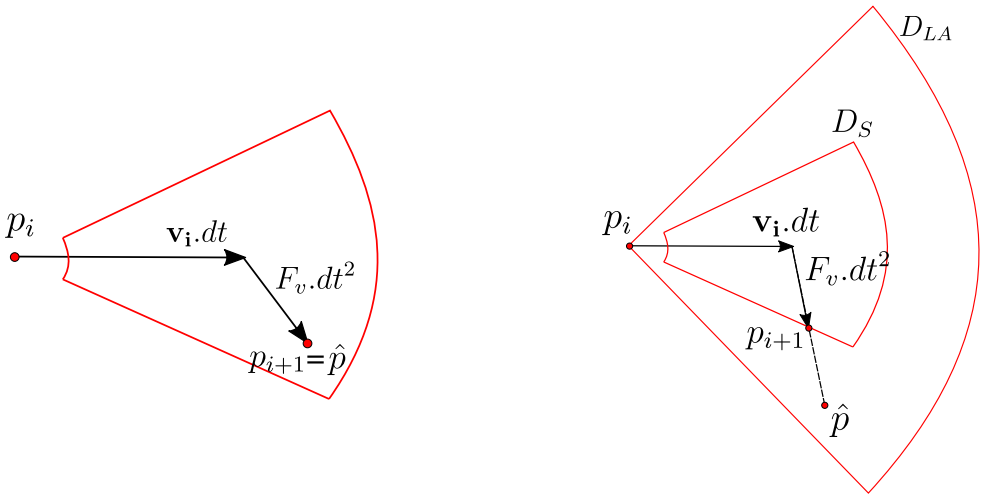
3.4.1 Physically-plausible search areas

In the following, we define the shape of the search areas D_S and D_{LA}, as well as explain how to evaluate occlusions in area D_C.



(a) Low velocity (b) Medium velocity (c) High velocity

Figure 3.8 – Impact of camera velocity on the single-frame search area.



(a) Single-frame search strategy.

(b) Look-ahead search strategy (over several frames).

Figure 3.9 – Local strategies

Single-frame search The locations which our camera can physically reach within a single frame define an area bounded by the range of possible forces $\mathbf{F}_v$ which can be applied to the camera. Here we propose to approximate this range of possibilities using a linear threshold and a rotational threshold (f_l and f_r respectively, see figure 3.8), *i.e.* enabling the camera to perform, at each frame, a range of changes in velocity and in direction. Obviously the range of changes are dependent on camera physical parameters such as previous velocity ($\mathbf{v}_i$) and damping factor (c). In an empirical way, we propose to express both changes as:

$$f_r = \frac{2\pi - \chi}{e^{g.\|\mathbf{v_i}\|dt}} + \chi \quad \text{, and} \quad f_l = 2h.\|\mathbf{v_i}\|dt \tag{3.3}$$

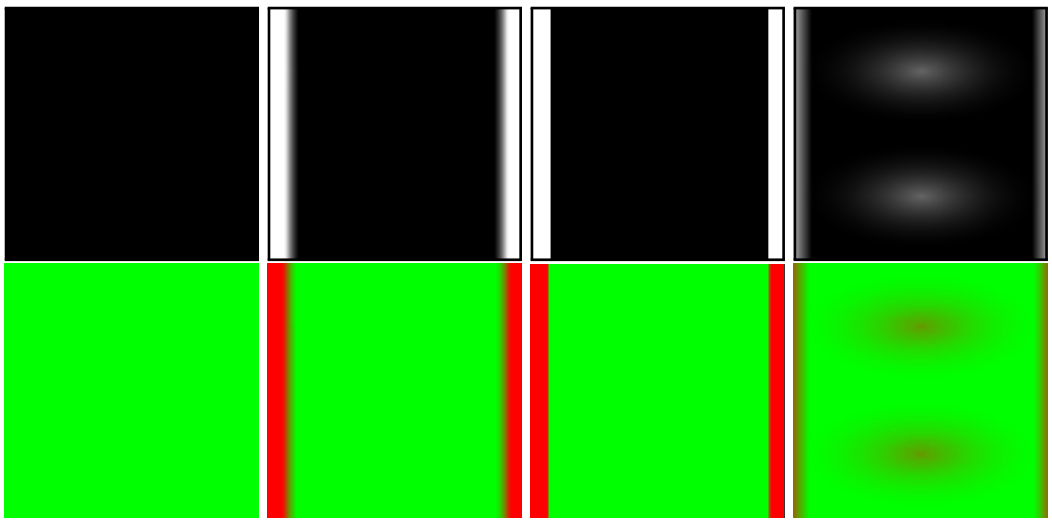
where f_r represents the angle threshold dependent on the speed, which is used to define the range of possibilities in angle $[-f_r/2, f_r/2]$. Constant χ is a constant (in radian) representing the angular span at high speed, and g is a constant defining how the angles evolves as a function of the camera velocity. f_l represents a relative distance threshold, also dependent on the speed, used to define the range of possibilities in positions as $\mathbf{p}_i + \mathbf{v}_i.dt + [-f_l/2, f_l/2]$. The constant h controls the linear freedom in speed. By playing with these parameters, one can easily control the camera reactivity to match specific requirements of an application.

With the current formulation, at low camera speeds (*e.g.* $\|\mathbf{v_i}\| = 0$), the threshold f_l is null. To handle such situations, we provide another constant ρ representing the radius in a way that $f_l \geq \rho$ in all cases. Thanks to this, we will keep searching at low camera speeds (see figure 3.8a) and provide a sufficient impulse as soon as a local solution is found. After this local search, if we have successfully found a less occluded location, we can update the camera position, *i.e.* $\mathbf{p}_{i+1} = \hat{\mathbf{p}}$.

Look-ahead strategy In case there is no better solution in D_S, we search a larger area D_{LA} by bounding locations which the camera can physically reach within a few frames, rather than just the next frame. We simply need to replace dt by $N dt$ in equation 3.3 ($N > 1$ is a user-defined value). After this new local search, if a solution is found, we can update the camera position, moving it in the direction of the best solution, while staying in D_S, *i.e.* $\mathbf{p}_{i+1}$ is the intersection of the edge of D_S and the segment $[\mathbf{p}(\mathbf{v}_i), \hat{\mathbf{p}}]$ (see Fig. 3.9b). Assuming that a solution has been found, we need to update the velocity $\mathbf{v}_{i+1}$, which could be done by simply deriving the camera's position. In practice, this velocity is dampened so that the camera stabilizes when there are no occlusions:

$$\mathbf{v_{i+1}} = \left(\frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{dt}\right)\left(1 - \frac{c}{m}dt\right)$$

Cut strategy The cut strategy is applied when no better solution can be found in D_S and D_{LA} and enables the camera to jump to another location. In such case we search in the area D_C, which is much larger than the current camera neighborhood. Indeed, as illustrated in figure 3.5, the **A-map** computation is only performed in a local area around the current camera position. In fact, recomputing the entire map (Toric surface) would be too inaccurate because of high distortion in the rendered buffers, and low precision around the camera position. Furthermore, the camera performs a continuous motion



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

Figure 3.10 – Examples of masks (top) and the impact on the **A-map** (bottom). (a) Empty mask, (b) flexible exclusion of viewpoints behind targets (with a gradient), (c) hard exclusion of viewpoints behind targets, (d) smooth repulsive mask avoiding regions behind the targets, top views and bottom views. Mask images (top) are surrounded by a box to highlight edge gradients.

in most cases, so computing the whole map is unnecessarily expensive. This D_C area is additionally pruned for camera locations with a $30°$ angle rule of the current camera location. This a very common film editing convention [TB09a] preventing jump-cuts.

To perform this search in D_C we propose to rely on ray-casting. We cast rays to the same sample points s as before, and only rely on these tests to compute an occlusion ratio (*i.e.* there is no occlusion prediction). As soon as a less occluded position is found we perform a cut by teleporting the camera, *i.e.* $\mathbf{p}_{i+1} = \hat{\mathbf{p}}$. Further, in this case, we reset the camera's velocity, *i.e.* $\mathbf{v}_{i+1} = \mathbf{0}$.

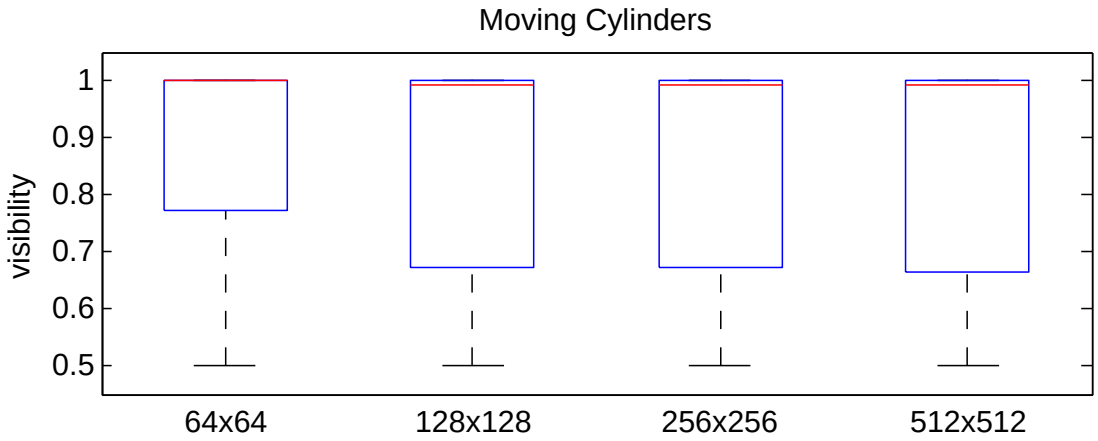
When no better solution has been found in these 3 searches, then we consider that the best option is to leave our camera particle follow its current path, *i.e.* $\mathbf{p}_{i+1} = \hat{\mathbf{p}} = \mathbf{p}(\mathbf{v}_i)$.

3.4.2 Style masks

While the camera moves autonomously to minimize occlusion, one might also want to avoid some viewpoints, depending on the targeted application. For example, in interactive applications with character control, one might prefer to not see the scene from

Table 3.2 – The set of default values we used to run our tests'

parameter	r	δ_w	γ_w	χ	g	h	ρ
value	5	0.6	0.5	10°	5	0.5	0.1



(a) Visibility performance for different maps resolutions (the higher, the better).

Size	64x64	128x128	256x256	512x512
FPS	197.09	130.04	88.80	18.96

(b) Frame rate for different map resolutions.

Figure 3.11 – Comparison of performances for different map sizes. Red: median value; Blue: 1st to 3rd quartile. The resolution impacts strongly the computational cost (b) but not the capacity to maintain visibility (a).

bellow or from above angles. For this purpose, we propose the notion of style masks on the **A-map** to influence the camera motion, or avoid specific areas (see examples in figure 3.10). We update the **A-map** using the following formulation:

$$A(x, y) = A(x, y) * Mask(x, y)$$

3.5 Results

Implementation We implemented our camera system within the Unity3D 2018 game engine. We compute our G-buffers through the Unity's integrated pipeline, while we perform our image processing stages (sections 3.3.1, 3.3.1, 3.3.2 and 3.4.2) through

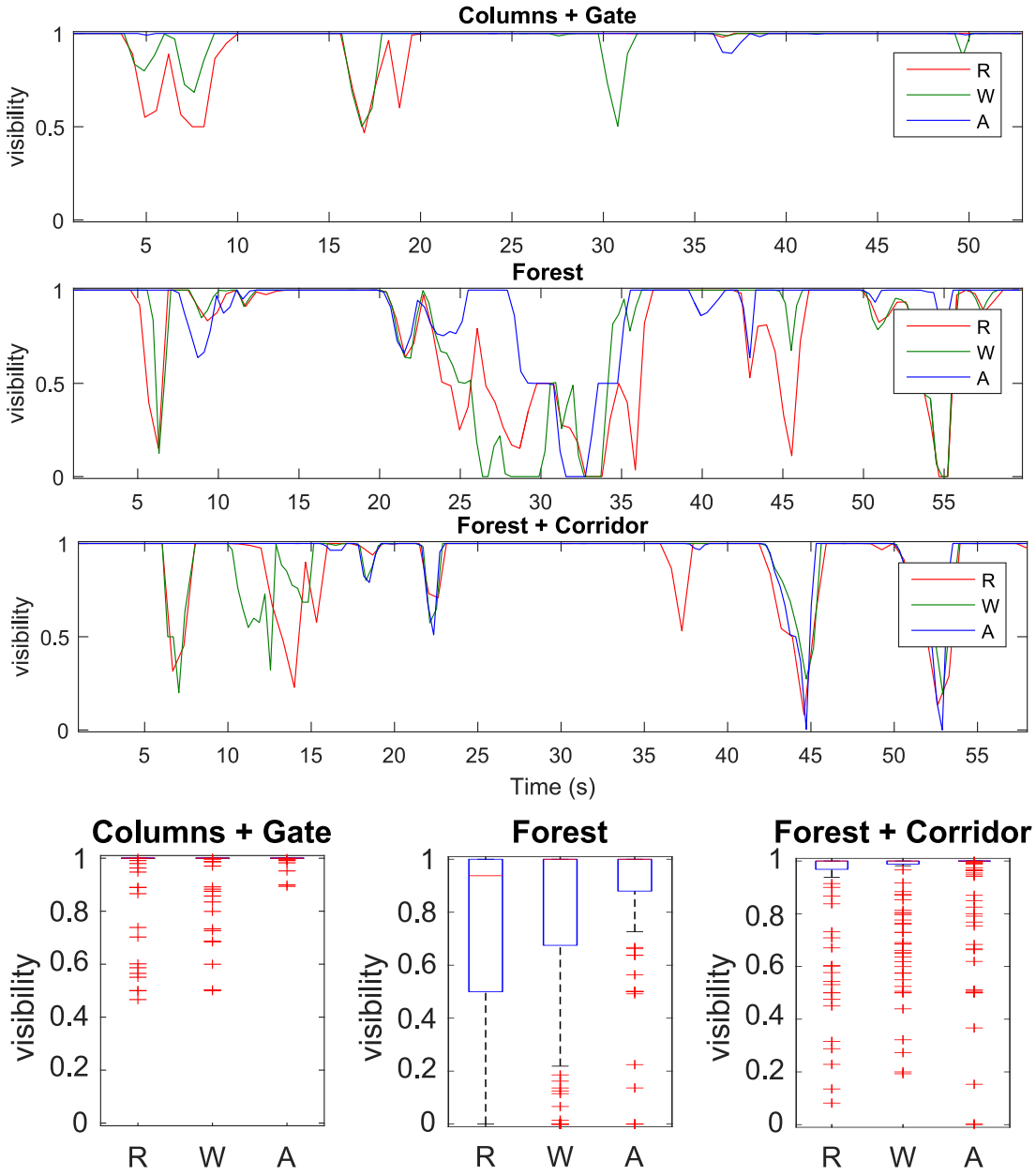
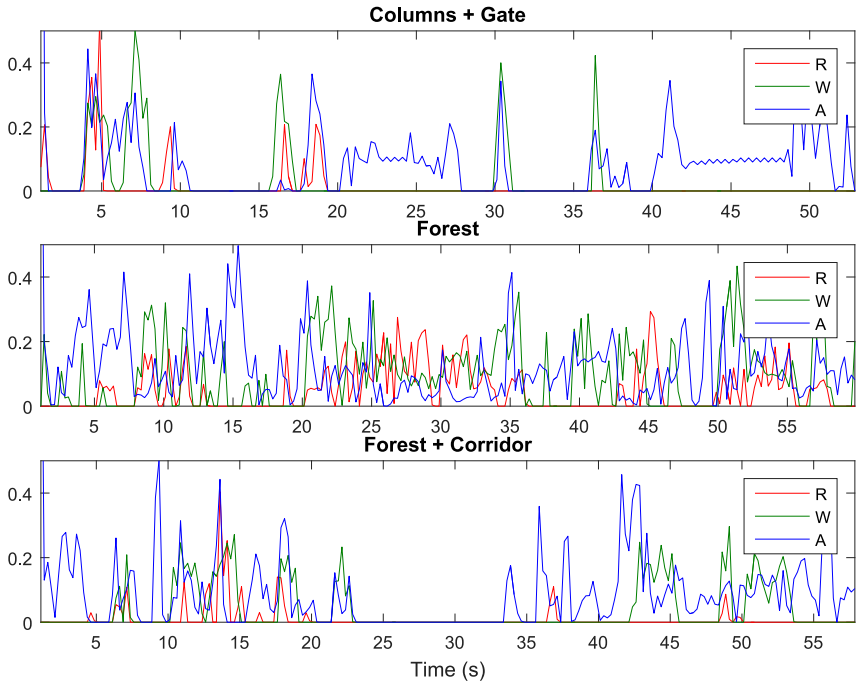
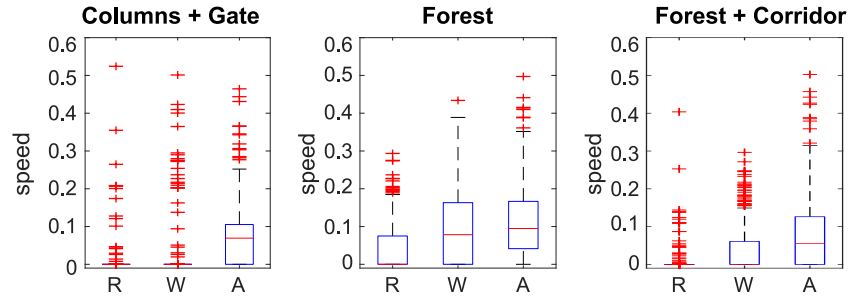


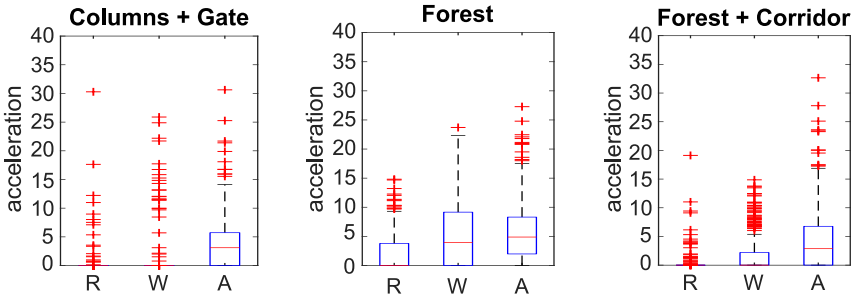
Figure 3.12 – Visibility comparison on 3 complex scenes, using the ray-cast method (R), our method with anticipation (A) or our method without anticipation (W). (top) Visibility over time (in s), (bottom) side-by-side comparison of performances. Red line: median value; Blue: 1st to 3rd quartile; Red crosses: outliers.

Unity Compute Shaders. All our results (detailed in section 3.5) have been processed on a desktop computer with an Intel Core i7-7820X CPU @ 3.60GHz and an NVIDIA Titan XP.





(a) Speed



(b) Acceleration

Figure 3.13 – Motion features on complex scenes, if using the ray-cast method (R), or our method with (A) or without (W) anticipation. (top) reported camera speed for 3 benchmarks, (bottom) speed (a) and (acceleration) distribution.

We evaluate our camera system along two main criteria: how much the targets are

	Each map update		Each frame	
Steps	Projections (**S-map**+ **V-map**)	**A-map**	Search	Camera update
mean	9.8	11.9	4	2.8
(st. dev)	(3.16)	(2.25)	(0.95)	(1.34)

Table 3.3 – Computation time in ms. We use 64x64 maps, 3 sample points per target (at each update, we recompute the maps for one sample on each target), and 50 sampled points in the search area.

visible (1 meaning that both targets are visible, 0 both targets are occluded), and how smooth the camera motions are. We compare the performance of our system with different sets of parameters and features *i.e.* changing the size of computed maps, and using (or not) our occlusion anticipation *vs.* a commonly used occlusion computation. To this end, we perform comparisons on 4 scene configurations (illustrated in the accompanying video) made of a simple scene and three complex scenes: (ii) a scene with a set of columns and a gate (**Columns+Gate**) which the target avatar goes through (iii) a scene with the avatar travelling a forest (**Forest**) and (iv) a scene with the avatar travelling a mix of forest and corridors with big walls (**Forrest+Corridor**).

Comparisons are performed as post-process to not influence the performance of the system. To provide a fair comparison between techniques, we measure the visibility degree on both targets by casting many rays (1 ray per vertex on each target). In a way similar, we evaluate the quality of camera motions by computing their derivatives, *i.e.* speed and acceleration, which provide a good indication of camera smoothness.

Impact of map resolution As a stress test, we ran our system on our simple scene configuration, where a pair of targets (spheres) is moving in a scene which we progressively fill with cylinders moving in random directions. We add up to 70 cylinders (4 per frame) until 47s of simulation; the whole simulation lasts about 60s. We ran this test with medium-size cylinders, or large cylinders. This allowed to evaluate the ability of our system to compute and enforce visibility by comparing the mean (actual) visibility over time, and the average frame rate. As shown in figure 3.11, decreasing the map resolution does not yield any noticeable loss in visibility enforcement, even for 64x64 maps. Conversely, there is a noticeable computational speedup. In all our following tests we rely on 64x64 maps.

As a second test, we ran our system on the three complex scenes and focused on the

system's performances when using a brute-force ray-casting to evaluate visibility (R), or using our computed maps while enabling (A) or disabling (W) the occlusion anticipation stage. When disabled (W), our **A-map** was computed as the averaged value in each **S-map**, *i.e.* we remove the use of our uncertainty function. Brute-force raycasting is very common in game engines, but its cost prevents the computation of our anticipation map with real-time performances; hence, in this case (R) we directly cast rays (to the $2r$ sample points) at the search stage (*i.e.* we compute no anticipation map). For all three techniques, we compare their cost, and ability to enforce visibility (figure 3.12) and to provide smooth camera motions (figure 3.13). We then provide a breakdown of the computational cost of our method (A) (table 3.3).

Impact of the visibility computation technique From our results, it appears clearly that with (A) or without (W) anticipation methods always improve visibility on targets compared to the ray-cast based method (R). Further, enabling the anticipation stage (A) provides an improvement compared to disabling it (W). Typically, in the *Forest* scene which is a scene specifically designed with a high density of occluders, the method with anticipation (A) shows the best performance (see figure 3.12). In the *Columns + Gate* scene, there is a clear benefit in using our anticipation step, especially at moments where the camera has to follow the target through the gate.

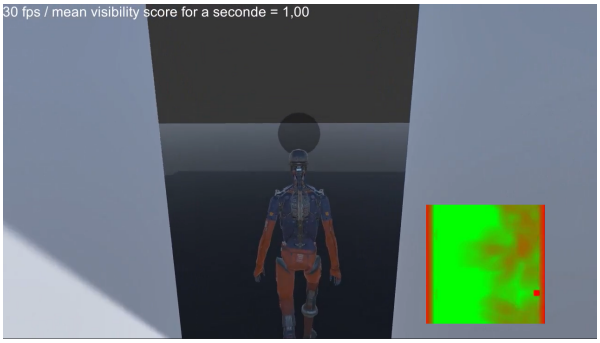
Motion smoothness Our motion strategies lead to smaller variations of velocity and acceleration (figure 3.13), while outliers can be due to either cuts, or our strategy at low camera speed, providing some impulse to the camera. Furthermore, when anticipation is not used ((R) and (W)), the acceleration remain lower, but at the cost of reducing the visibility.

Cost of the visibility evaluation As expected, casting rays (on the CPU) is much more expensive than computing our maps (on the GPU). In our tests, we experimentally chose $r = 5$ to enable a fair comparison of the system's performances (*i.e.* a fast-enough frame rate for the ray-cast based method (R)). As expected intuitively, the cost of computing our maps is linear in the number r of samples, while computing and fetching the anticipation map is made at a fixed cost, as we perform the search on the GPU (hence in parallel).

Walking to a column that will occlude the line of sight of the character.

The camera avoids the occlusion from the column by moving to the right of the character.

The character is going through a door.



The camera is not able to fully avoid the line of sight occlusion from the character's head.

Figure 3.14 – Results on an environment composed by columns and a doorway. The two targets are the robot character and its line of sight (*i.e.* black sphere). The map on the down right corner is the current anticipation map with a mask to avoid the toric extremities.

Computational cost breakdown As shown in table 3.3, the most expensive stage per frame is the computation of the **A-map**, then the projections of all **S-map**s and **V-map**s. The search and camera update are conversely inexpensive stages. To improve computational costs, we notice that, by tweaking parameters δ_W and γ_W, we can predict occlusion for a long-enough time window W. Doing so, we propose an optimization: to not update all $2r$ partial **A-map** at every frame, but 2 of them only (one per target). In other words, each map A_s^o would be re-computed every r frames. Moreover, we propose to not make an update at every frame, but instead to use a refresh rate matching our time window W. It will dictate when to make an update, *i.e.* every W/r seconds. In our tests, we make an update every 0.1 second (*e.g.* if $r = 5$, any map is updated every 0.5 second). We experimentally noticed that, for moderate values of r, the impact of this optimization on our system's visibility criteria is not noticeable.

Walking by a plant that will occlude the line of sight of the character.

The camera avoids the occlusion from the plant by going above the plant.

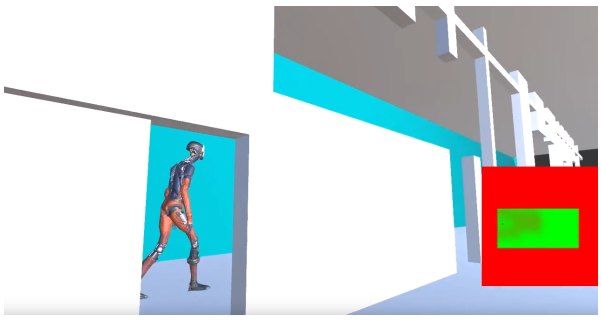
The character is going through a dense vegetation area.

The camera is not able to avoid the occlusion from the plant and will jump to an occlusion free position.

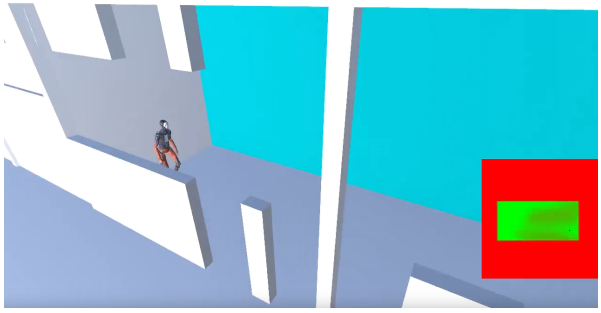
Figure 3.15 – Results on an environment composed by dense vegetation. The two targets are the robot character and its line of sight (*i.e.* black sphere). The map on the down right corner is the current anticipation map with a mask to avoid the toric extremities.

3.6 Discussion and Conclusion

We have highlighted the key features of our approach: low computational cost and improved visibility through anticipation. There however remains some limitations. First, there is an intrinsic limitation in the underlying Toric space representation, *i.e.* the camera must move at the surface of a moving torus, while there is no such restriction in the 6D camera space. We would like to investigate the provision of slight changes in the desired screen locations of targets to expand the range of motions; for instance we could also compute a slightly smaller and a slightly wider torus, leading to the computation of a 3D **A-map**. Typically, this would help to move the camera closer to the avatar, *e.g.* to avoid some occlusion at the expense of loosing composition. In our future work, we would also like to better exploit the depth information in the **S-map** to derive a camera



The camera is in a local minimum and if it does not choose to move, the target will be occluded by the wall.



The camera therefore cuts to an occlusion-free area to get out of the local minimum.

Figure 3.16 – Results on an environment composed of a corridor and a wall with holes. The target is the robot character. The map on the down right corner is the current anticipation map with a mask to avoid the toric extremities and the corridor interior. In this tracking task, the target walks forward through the corridor.

motion control over all Toric space parameters (φ, θ and α). Second, our approach is primarily designed to avoid occlusion, while simple secondary constraints can be added with style masks. More complex (and ideally dynamic) secondary constraints could be added by providing better representations. Finally, taking a step further, the Toric space we rely on is one possible model, allowing to enforce on-screen constraints; in the future, we would like to adapt our rendering+anticipation framework to perform our anticipation for 3D primitives defined around one or more targets, on which we could define more complex motions with physical constraints.

In this chapter, we proposed a real-time occlusion avoidance approach for virtual camera control. The system first computes an occlusion map by projecting occluders on a camera control surface (the Toric surface), and then exploits information on the velocity of the occluders vertices, to derive an anticipation map. The anticipation map is then exploited by a physical camera system to compute a new camera position minimizing occlusions. We compared our system with an elaborate ray-casting approach, and with our system in which anticipation was disabled. Results reported better performances both in terms of computational cost (compared to ray-casting), overall visibility and smooth motions both in terms of camera velocity and acceleration.

Here we mainly focused on occlusion avoidance. Due to the local nature of the approach, we only have a few means to characterize the trajectory quality that can only be smoothened in this work but not optimized for smoothing.

In the next chapter we will see how we address these problems by proposing a new camera animation space applied to a hybrid planing approach accounting for trajectory quality with dedicated visual properties.

CHAPTER 4

HYBRID APPROACH FOR CINEMATOGRAPHIC CAMERA PATH PLANNING

The first and foremost problem in this chapter is to identify what are the intrinsic characteristics of good camera motions. While film literature provides a thorough and in-depth analysis of what makes a qualitative viewpoint in terms of framing, angle to target, aesthetic composition, depth-of-field or lighting, the characterization of camera motions has been far less addressed. This pertains to specifics of real camera rigs (dollies, cranes) that physically limit the range of motions, and also the limited use of long camera sequences in movies except for Steadicam sequence shots. In addition, characteristics of camera motions in movies are strongly guided by the narrative intentions which need to be conveyed (*e.g.* rhythm, excitation, or soothing atmosphere) that are difficult to formalize.

In an attempt to transpose this knowledge to the tracking of targets in virtual environments, one can however derive a number of desirable cinematic characteristics such as *visibility* (avoiding occlusion of the tracked target and collisions with the environment), *smoothness* (avoiding jerkiness in trajectories) and *continuity* (avoiding large changes in viewing angles and distances to target). In practice, however, these characteristics are often contradictory: avoiding a sudden occlusion requires a strong acceleration, or an abrupt change in angle. Furthermore, the computational cost of evaluating visibility, continuity and smoothness along trajectories limits the possibility of evaluating many alternative camera motions.

Existing work have either addressed the problem using global motion planning techniques typically based on precomputed roadmaps [NO04]; [Osk+09]; [JLC20], or local planning techniques using ray casting [RU14] and shadow maps for efficient visibility computations [HHS01]; [CNO12]. While global motion planning techniques excel at ensuring visibility given their full prior knowledge of the scene, local planning techniques

excel in handling strong dynamic changes in the environment. The main bottleneck of both approaches remains the limited capacity in evaluating at run time the cinematic properties along a camera motion or in the local neighborhood of a camera position.

Our approach builds on the idea of performing a mixed local+global approach by exploiting a finite-time horizon that is large enough to perform a global planning, yet efficient enough to react in real-time to sudden changes. This sliding window exploits recent hardware raycasting techniques to enable the real-time evaluation of thousands of camera motions. As such, our approach draws inspiration from Model Predictive Control techniques [SM98] by optimizing a finite time-horizon, only implementing the current time slot and then repeating the process on the following time slots.

To implement this approach, we make the hypothesis that the target object is controlled by the user through interactive inputs. Its motions and actions can therefore be predicted within a short time horizon h. Our system comprises 2 main stages, illustrated in Figure 4.1. In the first stage, we predict the motion of the target over our given time horizon h by using the target's current position (at time t_i) and the user inputs. We then select an ideal camera position at time $t_i + h$ and propose to define the *camera animation space* as all camera animations that link the current camera position (at time t_i), to the ideal camera location (at time $t_i + h$) and to sample the animation space as a collection of smooth camera animations. In the second stage, we perform an evaluation of the quality of the camera animations in this animation space by relying on hardware raycasting techniques and select the best camera animation. In a way similar to Motion Predictive Control [SM98], we then apply part of the camera animation and re-start the process at a low frequency (4 Hz in our case) or when a change in the user inputs is detected. Finally, to better adapt the camera animation space to the scene topology (*e.g.* cluttered environments *vs.* open environments), we dynamically update a scaling factor on the animation space. As a whole this process allows generating a continuous and smooth camera animation which enables the real-time tracking of a target object in fully dynamic and complex environments.

Our contributions are:

— the design of a camera animation space as a finite time horizon space in which to express a range of camera trajectories;

— an efficient evaluation technique using hardware ray casting;

— a motion predictive control approach that exploits the camera animation space to generate real-time cinematic camera motions.

4.1 Overview

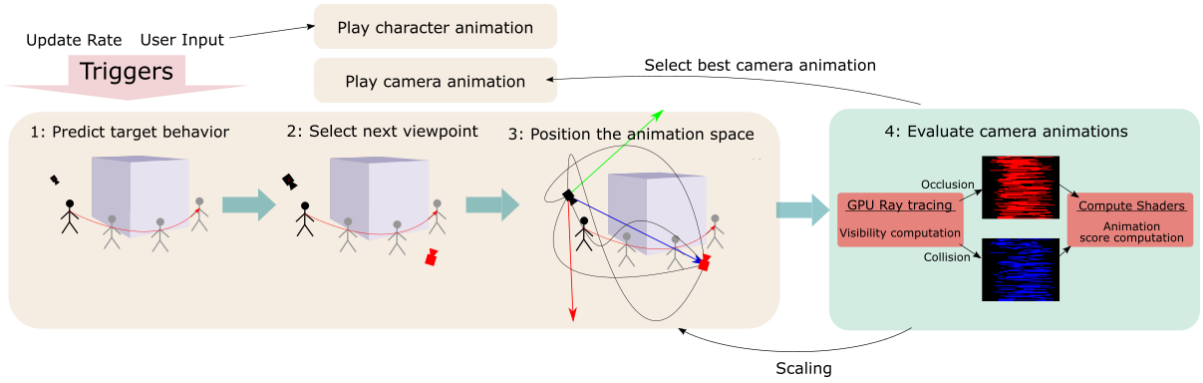


Figure 4.1 – System overview: the orange box represents the CPU part of the system; the green box represent the GPU part of the system

Our system aims at tracking in real-time a target object traveling through a dynamic 3d environment by generating series of smooth cinematic camera motions. In the following, we will present the construction of our camera animation space (Section 4.2) and then detail the evaluation of camera animations using hardware ray casting (Section 4.3). We will then show how the camera animation space can be dynamically recomputed to adapt to the characteristics of the scene topology (cluttered *vs.* open environments) and how this adaptation improves our results (Section 4.4).

4.2 Camera animation space

We propose the design of a *Camera Animation Space* as a relative local frame defined by an initial camera configuration $\mathbf{q}_{start}$ at time t_i and final camera configuration $\mathbf{q}_{goal}$ at time $t_i + h$ (see Figure 4.2). This local space defines all the possible camera animations that link $\mathbf{q}_{start}$ at time t_i to $\mathbf{q}_{goal}$ at time $t_i + h$. Our goal is to compute the optimal camera motion within this space considering a number of desired features on the trajectory (*e.g.* smoothness, collision and occlusion avoidance along the camera animation, viewpoint preferences and cinematic properties).

We propose to follow a 3-step process: (i) anticipate the target's behavior (*i.e.* its next positions) within the given time horizon, (ii) choose a goal camera viewpoint from which to view the target at the end of the time horizon, and (iii) given this goal view-

Table 4.1 – Notations used in the paper

H^i	Time horizon for iteration i (between times t_i and $t_i + h$)
$\mathbf{B}^i(t)$	Target behavior (predicted position) at time $t \in H_i$
$\mathbf{V}^i$	Set of preferred viewpoints at time $t_i + h$
$\mathbb{Q}^i$	Camera animation space for horizon H^i
M^i	Transform matrix of the camera animation space, for H^i
$\mathbf{q}^i_j(t)$	3D position in camera animation $\mathbf{q}^i_j \in \mathbb{Q}^i$, at time $t \in H^i$
$\mathbf{q}^i_{start}$	Starting camera position. $\mathbf{q}^i_j(t_i) = q^i_{start},\ \forall\ (i,j)$
$\mathbf{q}^i_{goal}$	Goal camera position. $\mathbf{q}^i_j(t_i + h) = q^i_{goal} \in \mathbf{V}^i,\ \forall\ (i,j)$
$\dot{\mathbf{q}}(t)$	Tangent vector of a camera track at time t
$D^i_j(t)$	The camera view vector at time t
$(\mathbf{x}, \mathbf{y})$	angle between two vectors $\mathbf{x}$ and $\mathbf{y}$
$G(x, \sigma)$	Gaussian decay, equals to $e^{-x^2/(2\sigma^2)}$
$E(x, \lambda)$	Exponential decay, equals to $e^{-x/\lambda}$

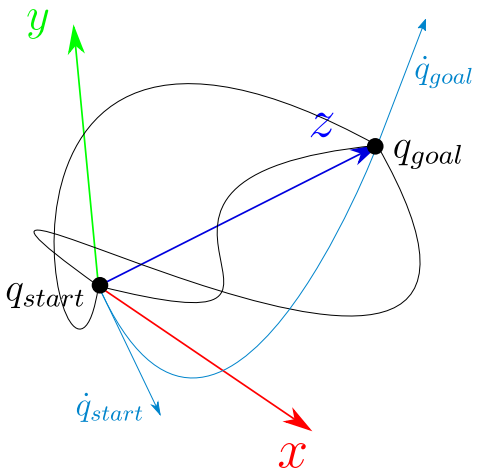


Figure 4.2 – Representation of our animation space and its local transform

point, and the current one, build and evaluate the space of possible camera animations between them using our camera animation space.

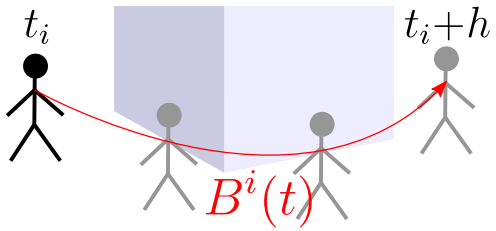


Figure 4.3 – Representation of the target's behavior curve at iteration i

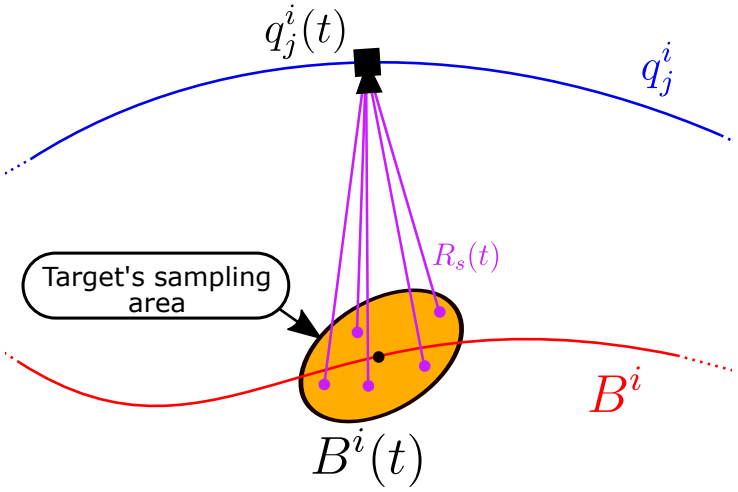


Figure 4.4 – Ray launched from the camera toward the target's sampling area at time t

4.2.1 Anticipating the target behavior

We here make a strong assumption that we can anticipate the next positions of the tracked target within a time horizon H^i. We consider H^i begins at time t_i and has a constant user-defined duration of h seconds. Moreover, we consider that the target's behavior will be consistent over the whole horizon H^i (a behavior being a motion among walking, running, turning, jumping). In our implementation, we consider the target as a rigid body with a speed, acceleration, and behavior. We then simulate the target's motion over time horizon H^i (avoiding collisions with obstacles) and store all simulated positions over time. We rely on PhysicsScene tool from Unity to perform the physical simulation. With this anticipation, we account for the scene geometry which might influence future user inputs. We then refer to the anticipated positions as the target simulated behavior, expressed in the form of a 3d animation curve $B^i(t)$ with $t \in H^i$ (see Figure 4.3). Note that one may use another technique to anticipate the target behavior. As long as it outputs a 3d animation curve over time, it will not change the overall

workflow of our camera system.

4.2.2 Selecting a goal viewpoint

We now make a second assumption that the user defines a set of viewpoints to portray the target object. By default, one might use a list of stereotypical viewpoints in movies such as 3-quarter front and back views, side views, or bird eye views. These viewpoints are sorted by order of preference in a priority queue $\mathbf{V}$ (order can be fixed by the user, randomly chosen, or scene dependent). Each viewpoint is defined as a 3d position in spherical coordinates (d, ϕ, θ), in the local frame of the target's configuration, where (ϕ, θ) defines the vertical and horizontal viewing angles, and d the viewing distance.

Considering all viewpoints are in $\mathbf{V}$, we pop viewpoints by order of priority. We propose to stop as soon as a viewpoint is promising enough, *i.e.* at time $t_i + h$ neither the target will be occluded from this viewpoint, nor the camera will be in collision with the scene geometry. We then refer to this selected viewpoint as the goal viewpoint $\mathbf{q}_{goal}$.

Knowing the current camera viewpoint $\mathbf{q}_{start}$ (at time t_i) and this goal viewpoint $\mathbf{q}_{goal}$ (at time $t_i + h$), we can define our camera animation space that we further sample and evaluate to select the optimal camera animation.

4.2.3 Sampling camera animations

Given the target behavior to track represented as a curve $B^i(t)$ and the two key viewpoints $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$, we propose to sample a large set of camera animations between the key viewpoints. We will hereafter note this stochastic set of camera animations as $\mathbb{Q}^i$, and a sampled camera animation as $\mathbf{q}^i_j$, where j is the sample index.

Two requirements should be considered on this sampled space: (i) sampled camera animations should be as-smooth-as-possible, *i.e.* with low jerk, and (ii) the sampled animation space should enforce continuity between successive horizons. To do so, we propose to encode each sampled camera animation as a cubic spline curve on all 3 camera position parameters, as they offer C^3 continuity between key-viewpoints. In practice, we make use of Hermite curves which eases the sampling by randomly selecting tangent vectors to the spline curve at start and end positions. C^1 continuity between successive Hermite curve portions is enforced by aligning both positions and tangents at connecting positions.

In practice, we propose for each camera animation to complement the starting and the goal camera positions $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ by two tangents, *i.e.* the camera velocities $\dot{\mathbf{q}}_{start}$ and $\dot{\mathbf{q}}_{goal}$ (figure 4.2). To offer a good coverage of the whole animation space, we use a uniform sampling of these tangents in a sphere of radius r (in our tests, we used $r = 5$). The number of sampled animations is left as a user-defined parameter. An evaluation of results for different values is provided in Section 4.5.2.

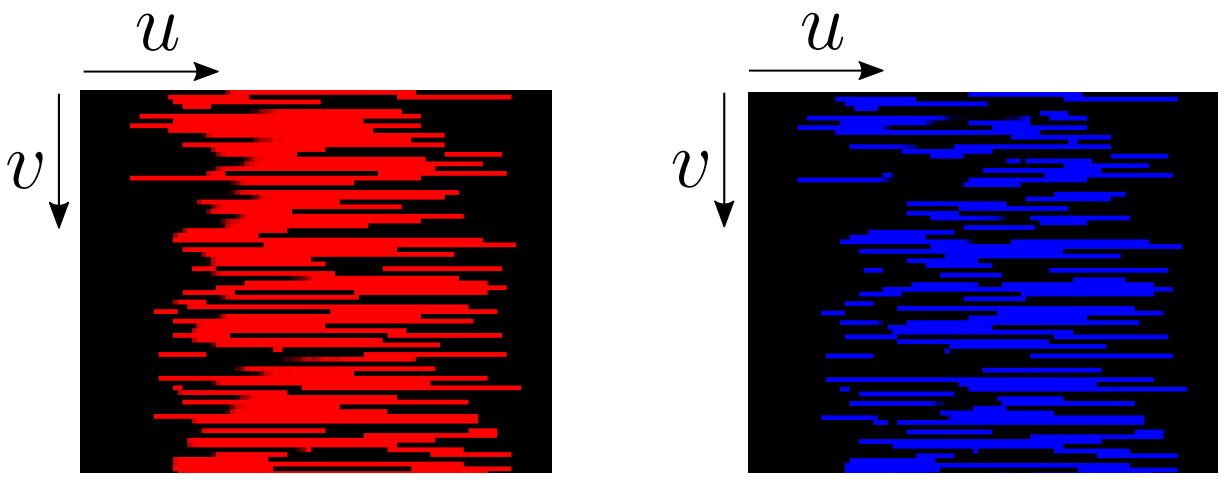
The frequent recomputation of the tangent sampling and camera path construction has two drawbacks: its computational expense and the lack of stability over time. To avoid the recomputation, we propose to precompute a graph of uniformly sampled camera animations, in an orthonormal coordinate system (as illustrated in figure 4.2). In this system, $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ have coordinates $(0, 0, 0)$ and $(0, 0, 1)$ respectively. Then, for any horizon H^i, we apply a proper 4×4 transform matrix M^i to align the graph onto the computed viewpoints $\mathbf{q}^i_{start}$ and $\mathbf{q}^i_{goal}$. It is worth noting that in M^i the 3d translation, 3d rotation and the scaling on the z axis will lead this axis to match the vector $(\mathbf{q}^i_{goal} - \mathbf{q}^i_{start})$. Two parameters remain free: the scaling for the other two axes (x and y). As a first assumption we could use the same scaling as for z. However, we will further explain how to choose a better scaling in section 4.4, in order to adapt the sampled space to the scene geometry.

4.3 Evaluating camera animations

In the first stage, we have computed a set of camera animations $\mathbb{Q}^i$, that can portray the target objects' behavior within time horizon H^i. We now need to select one of these animations as the one to apply to the camera.

4.3.1 Evaluating camera animation quality

Our second stage is devoted to evaluate the quality of all animations and selecting the most promising one efficiently. In the following, we will first detail our evaluation criteria, before focusing on how we perform this evaluation. A camera animation that portrays the motions of a target object should follow a number of requirements, among which the most important are: avoid collisions with the scene and enforce visibility on the target object, while offering a smooth series of intermediate viewpoints to the viewer. To evaluate how well these requirements are satisfied along a camera animation



(a) Occlusion (b) Collision

Figure 4.5 – Example of a part of a Visibility data encoding texture; Black = the target is visible from the camera, Red = the target is occluded or partially occluded from the camera, Blue = the camera is inside the scene geometry.

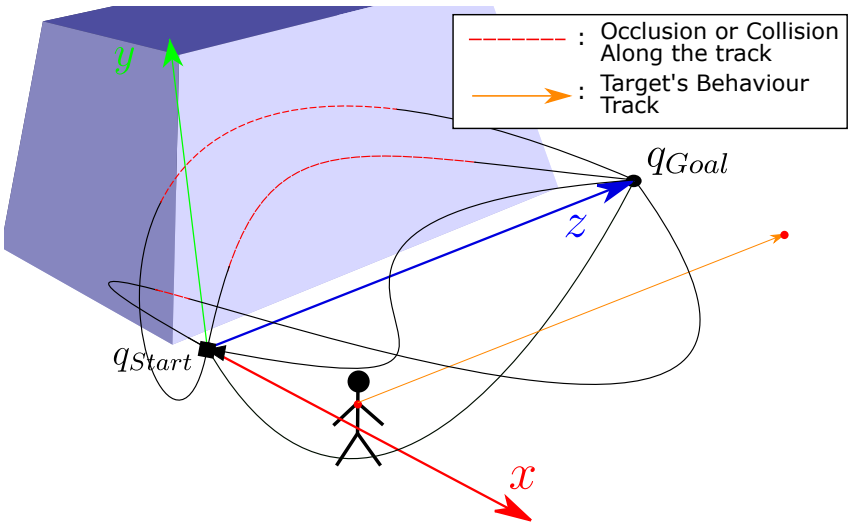


Figure 4.6 – Representation of the positioned animation space where parts of some trajectories collide with the scene geometry.

$\mathbf{q}_j^i$, we propose to rely on a set of costs $C_k(t) \in [0, 1]$:

Occlusions and Collisions To evaluate how much the target object is occluded from a camera position $q_j^i(t)$, we rely on ray casting as recommended by [RU14]. We first approximate the target's geometry with a simple abstraction (in our case it is a sphere

of the size of the character's upper body). Second we sample a set of points $s \in [0, N]$ on this abstraction, which we position at the object's anticipated position $B^i(t)$. Third, we launch a ray from the camera to each point s (see figure 4.4). We note $R_s(t)$ the result of this ray cast. We use the same ray to also evaluate if the camera is in collision (*i.e.* inside another object of the scene), by setting its value as:

$$R_s(t) = \begin{cases} 0 & \text{if Visible} \\ 1 & \text{if Occluded} \\ 2 & \text{if Collided} \end{cases} \tag{4.1}$$

We distinguish a collision from a simple occlusion as follows. By looking at the normal at the hit geometry, we know if the ray has hit a back face or a front face. When the ray hits a back face, $q^i_j(t)$ must be inside a geometry, hence we consider it as a camera collision. Conversely, when the ray hits a front face, $q^i_j(t)$ must be outside a geometry. If the ray does not reach s, we consider s as occluded, otherwise we consider it as visible.

Knowing $R_s(t)$, we define our collision and occlusion costs as:

$$C_o(t) = \frac{1}{N} \sum_{s=0}^{N} \begin{cases} 1 & \text{if } R_s(t) = 1 \\ 0 & \text{Otherwise} \end{cases} \tag{4.2}$$

and

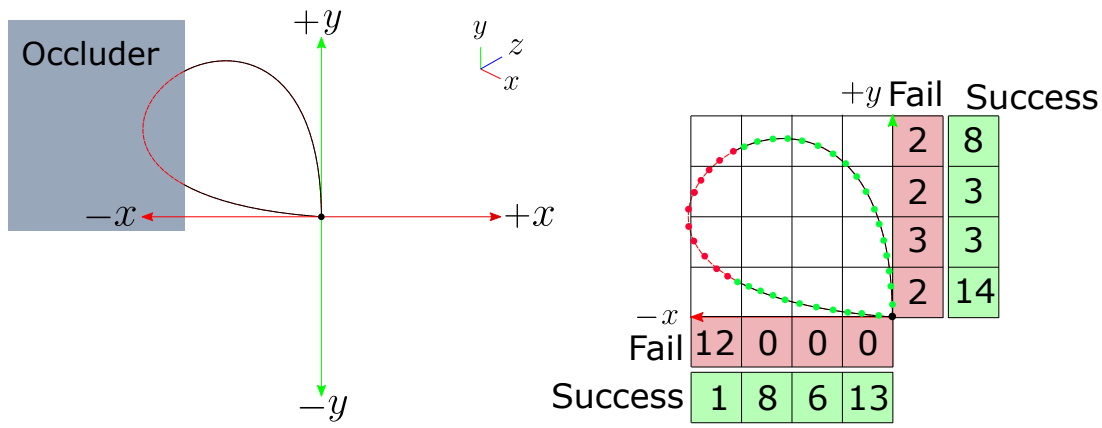
$$C_c(t) = \frac{1}{N} \sum_{s=0}^{N} \begin{cases} 1 & \text{if } R_s(t) = 2 \\ 0 & \text{Otherwise} \end{cases} \tag{4.3}$$

In our tests, we used $N = 20$.

Minimizing visual changes A smooth camera motion is a motion that avoids sudden changes in visual properties (distance to target and angle to target). We therefore propose an addition metric to evaluate how much the viewpoint changes over time. We split this evaluation into two distinct costs: one on the camera view angle, and one on the distance to the target object. Costs are evaluated for each time step δt.

Let us denote $D^i_j(t)$ the view vector connecting the target object to the camera computed as:

$$D^i_j(t) = B^i(t) - q^i_j(t)$$



(a) 2D view of an occluded track from the animation space

(b) Computation of the fail and success histogram on each axis

Figure 4.7 – Projection of the success and fail of one camera animation on the four axis of resolution $R = 4$. (a) Collision and occlusion detection (b) Enumeration and projection of the success and fail samples on the axis.

From this view vector, we define the view angle change as:

$$C_{\Delta_{\phi,\theta}}(t) = \frac{(D^i_j(t), D^i_j(t+\delta t))}{\pi} \tag{4.4}$$

In a way similar, we propose to rely on a squared distance variation, defined as:

$$\Delta d(t) = (\|D^i_j(t)\| - \|D^i_j(t+\delta t)\|)^2$$

We then define a cost on this distance change which we further normalize as:

$$C_{\Delta d}(t) = 1 - E(\Delta d(t), \lambda) \tag{4.5}$$

where E is an exponential decay function, for which we set parameter λ to 10^{-4}.

Preferred range of distances One side effect of the above costs is that for large distances, changes on the view angle and distance will be less penalized. In turn, this will favor large camera animations. It is worth noting that, in the same way, placing the camera too close to the target object is also not desired in general. We should then penalize both behaviors. To do so, we propose to introduce a last cost, aimed at favoring camera animations where the camera remains within a prescribed distance range $[d_{min}, d_{max}]$.

We formulate this cost as:

$$C_d(t) = \begin{cases} 1 & \text{if } \|D_j^i(t)\| \notin [d_{min}, d_{max}] \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

In our tests, we used $[d_{min}, d_{max}] = [0.4, 1]$.

4.3.2 Selecting a camera animation

In a first step, we define the total cost of a camera animation as a weighted sum of its single-criteria costs integrated over time:

$$C = \sum_k w_k . \left[\int_{t_i}^{t_i+h} C_k(t)\ G(t - t_i, \sigma)\ dt \right] \tag{4.7}$$

where $w_k \in [0, 1]$ is the weight of criterion k. G is a Gaussian decay function, where we set standard deviation σ to the value of $h/4$. We also slightly tune the decay to converge towards 0.25 (instead of 0). This way, we give a higher importance to the costs of the beginning of the animation, yet still considering the end. Indeed, as in motion predictive control, our assumption is that the camera will only play the first part of it (10% in our tests), while the remaining part still brings a long term information on what could be a good camera path. In our tests, typical weights are $w_o = 0.4$, $w_c = 0.2$, $w_d = 0.12$, $w_{\Delta_{\phi,\theta}} = 0.04$, $w_{\Delta d} = 0.04$. We compute the total cost for any camera animation $q_j^i \in \mathbb{Q}^i$ by discretizing the time integral (details are given in the next section). We hereafter refer to this total cost as C_j^i.

In a second step, we propose to choose the most promising camera animation for time horizon H^i, denoted as $\mathbf{q}^i$, as the one with minimum total cost, *i.e.* :

$$\mathbf{q}^i = \arg\min_j C_j^i \tag{4.8}$$

4.3.3 GPU-based evaluation

We have presented our evaluation metric on camera animations. However, some costs are expensive to compute. In particular, occlusion and collision testing requires to trace numerous rays (*i.e.* N rays, for many time steps, for hundreds of camera animations). It is worth noting that many of our computations are independent and can

therefore be performed in parallel. Similarly, the evaluation of a cost at discretized time steps along a given animation are also independent. Hence, we propose to cast our evaluation of single costs into a massively-parallel computation on GPU.

We design our system in a way that we only need to send the animation space (in orthonormal coordinate system) once to the GPU. Then, when we need to reposition the camera animation space, for horizon H^i, we simply update the 4×4 transform matrix M^i. And, from this data, one can straightforwardly compute any camera position $\mathbf{q}_j^i(t)$ for any time t.

Second, for occlusion and collision computations, we propose to rely on the recent RTX technology allowing to perform real-time ray casting requests on GPU. To discretize the time integral of our costs, we run $\frac{h}{\delta t}$ thread per track (each thread corresponds to a discretized time step and are executed in parallel on GPU). Each thread launches N rays, one per sample s picked onto the target object. In our tests, we use $N = 20$, and 100 threads per camera animation. In turn, $\delta t = \frac{h}{100} = 0.05$. The result of these computations are stored into a 2D texture (as shown in figure 4.5), where the texture coordinates u and v map to one time step t and one animation of index j, respectively. Occlusion and collision costs are stored into two different channels.

Third, we rely on a compute shader to compute all other costs, and combine them with occlusion and collision costs. This shader uses one thread per camera animation. It stores the total cost of all animations into a GPU buffer, finally sent back to CPU where we perform the selection step.

4.4 Dynamic Trajectory Adaptation

Until now, we have considered a nominal situation where we evaluate the animation space and select one camera animation for one given time horizon H^i. We now need to consider two other requirements. First, the camera should be animated to track the target object for an unknown duration, larger than h. Changes in the target behavior may also occur, due to interactive user inputs. Second, for any horizon H^i, some camera animations could be in collision with the scene, or the target could be occluded. This would prevent finding a proper animation to apply. In other words, the space of potential camera animations should be influenced by the surrounding scene geometry. Hereafter, we explain how we account for these requirements.

4.4.1 User inputs and interactive update

We here assume the camera is currently animated along a curve $\mathbf{q}^i$. We then need to compute a new camera animation for a time horizon H^{i+1} in two cases. First, when the target's behavior is changed (*i.e.* the user input has changed). This event indeed breaks the validity of the currently played animation for future time steps. Second, it appears reasonable to consider the target behavior, collision and occlusion information will be less and less reliable as time advances. In a way similar to motion-predictive control, we then compute an animation for an anticipated horizon of length h, but only play the first steps, to account for possible dynamic collisions and occlusions. The duration of these first steps is specified as a user-defined ratio of progress along animation $\mathbf{q}^i$. In our tests we used a horizon length $h = 5$ seconds and a ratio of progress of 10%. In turn, the new horizon generally starts at $t_{i+1} = t_i + 0.1h$, while we set $\mathbf{q}^{i+1}_{start} = \mathbf{q}^i(t_{i+1})$.

When an update is required (behavior change, or ratio of progress reached), we recompute a new camera animation for the next horizon H^{i+1}: we select a new goal viewpoint (*i.e.* $\mathbf{q}^{i+1}_{goal}$) and update the transform matrix (*i.e.* M^{i+1}) to position the camera animation space $\mathbb{Q}^{i+1}$. We then evaluate all camera animations in $\mathbb{Q}^{i+1}$.

Animation transitions To enforce continuity between animation q^i and the animation $\mathbf{q}^{i+1}$ that is to be selected we rely on an additional cost designed to favor a smooth transition between consecutive animations. This cost penalizes abrupt changes when transitioning between two camera animation curves. Our idea is to penalize a wide angle between the tangent vector to camera animation $\mathbf{q}^i$ and the tangent vector to animation $\mathbf{q}^{i+1}_j \in \mathbb{Q}^{i+1}$, at connection time t_{i+1}. We write this cost as:

$$C_{i,i+1}(j) = \frac{(\dot{\mathbf{q}}^i(t_{i+1}), \dot{\mathbf{q}}^{i+1}_j(t_{i+1}))}{\pi} \tag{4.9}$$

We then rewrite the selection of camera animation q^{i+1} as:

$$\mathbf{q}^{i+1} = \arg\min_j \left[C^{i+1}_j + w_{i,i+1}\, C_{i,i+1}(j)\right] \tag{4.10}$$

where $w_{i,i+1}$ is the weight of the transition cost, in our test we use $w_{i,i+1} = 0.2$.

4.4.2 Adapt to scene geometry

Different scene geometries obviously impose different constraints on the camera animations and a single sampled camera animation space may not enough to tackle all situations. For example, cluttered environment such as corridors would ideally require dedicated samples. In fact, our camera animation space should exhibit as few collisions and occlusions as possible, while still covering as much as possible the free space between the target behavior and the scene geometry. To address this problem, rather than recomputing a new sampling of camera trajectories dedicated to these specific situations, we propose a technique to dynamically adapt our camera animation space to the scene geometry.

To do so, while we evaluate the quality of camera animations for a horizon H^i, we analyze how many collisions and occlusions occur. This informs us if the free space is well covered or not. We then propose to dynamically rescale the camera animation space to make it grow or shrink in the next time horizon H^{i+1}. This rescaling applies when we update the transform matrix M^{i+1}, and on the x and y axes only. It is worth noting that the free space might not be symmetrical around the target behavior (as illustrated figure 4.6 where the free space is larger on the left than on the right of the target). The same applies to the free space above or below the target. Consequently, our idea is to compute four scale values, on all four directions $\{-x, +x, -y, +y\}$ along the axis of the camera animation space. For any camera position along a camera animation, we then apply either two of them, depending on the sign of the position's x and y coordinates in the non-transformed animation space.

To compute this scaling we first leverage the occlusions and collisions evaluation to store additional information: we count fails and successes along each axis. We consider a launched ray along a camera animation (*i.e.* from the camera position at a given time step) as a fail if it is marked as occluded or collided, and as a success if not. Second, we store this information in height arrays: for each half-axis (*e.g.* $+x$ or $-x$), we count successes in one array, and fails in another array. We further discretize this half-axis by using a given resolution R and output two histograms of fails and successes (as illustrated in figure 4.7). Note that R here defines the scale precision on each axis. At last, we use both histograms to compute the new scale to apply. We compute the indices i_f and i_s of the medians of both arrays (fails and successes, respectively). By comparing them, we define how much we should rescale animations along this half-axis. If $i_s < i_f$, we consider that there are too many fails, and multiply the current scale by i_f/R to

shrink animations. Otherwise, we consider the free space is not covered enough, and apply a passive inflation to the current scale. The aim of this inflation is to help return to a maximum scale value, when the surrounding geometry allows for large camera animations.

4.5 Implementation and Results

4.5.1 Implementation

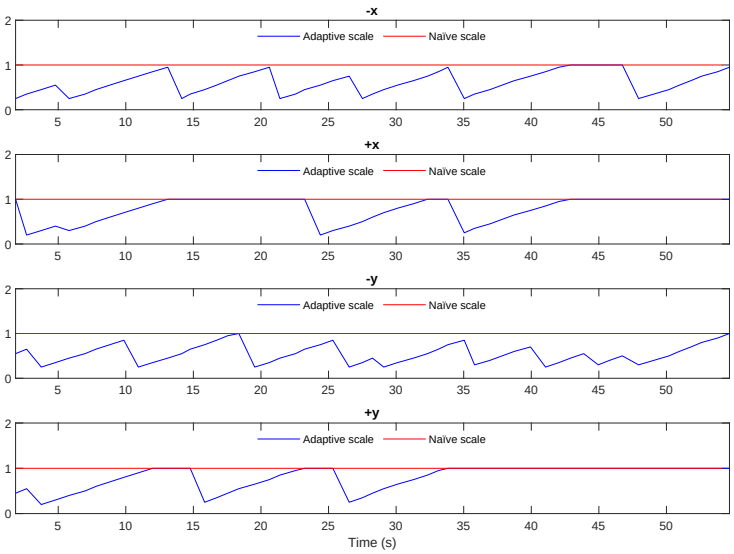
We implemented our camera system within the Unity3D 2019 game engine. We compute our visibility and occlusion textures through ray tracing shaders provided with Unity's integrated pipeline and perform our scores for all sampled animations and time steps through Unity Compute Shaders. All our experimentation (detailed in section 4.5.2) have been performed on a laptop computer with an Intel Core i9-9880H CPU @ 2.30GHz and an NVIDIA Quadro RTX 4000.

4.5.2 Results

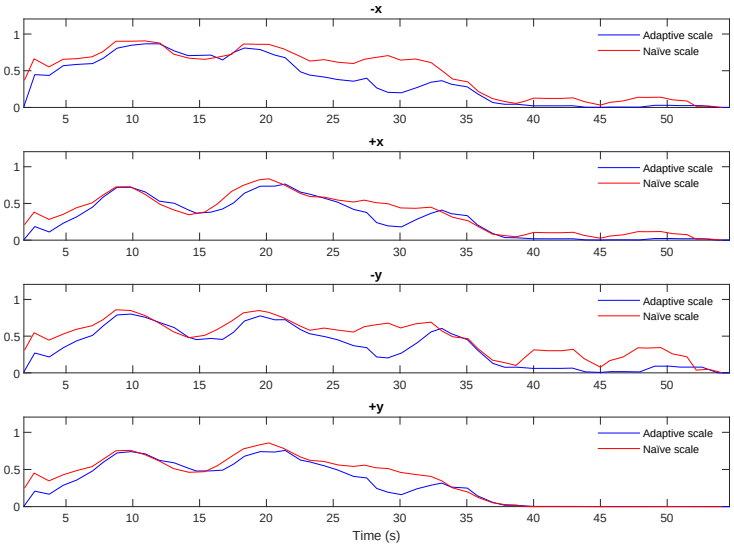
We split our evaluation into three parts. We first validate our adaptive scale mechanism. Then, we evaluate the robustness of our system, by comparing its performances when using a different number or set of reference camera animations. We finally validate the ability of our system mixing local and global planning approaches to outperform a purely local camera planning system. To do so, we compare results obtained with our system and the one presented in the previous chapter, on the same test scenes.

To validate our adaptive scale, we study its impact on the quality of the animation space. For the other evaluations, we compare camera systems using the two main criteria: how much the camera maintains the visibility on the target object and how smooth camera motions are. We compute visibility by launching rays onto the target object, and calculate the ratio of rays reaching the target. A ratio of 1 (respectively 0) means that the target is fully visible (respectively fully occluded). When relevant, we additionally provide statistics on the duration of partial occlusions. We then compare the quality of camera motions through their time derivatives (speed, acceleration and jerk), which provide a good indication of motion smoothness.

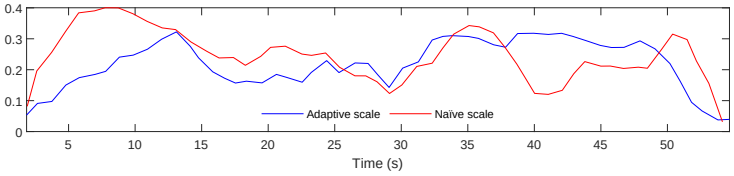
Our comparisons have been performed within 4 different scenes (illustrated in the



(a) Scale on each half-axis (-x, +x, -y, +y).

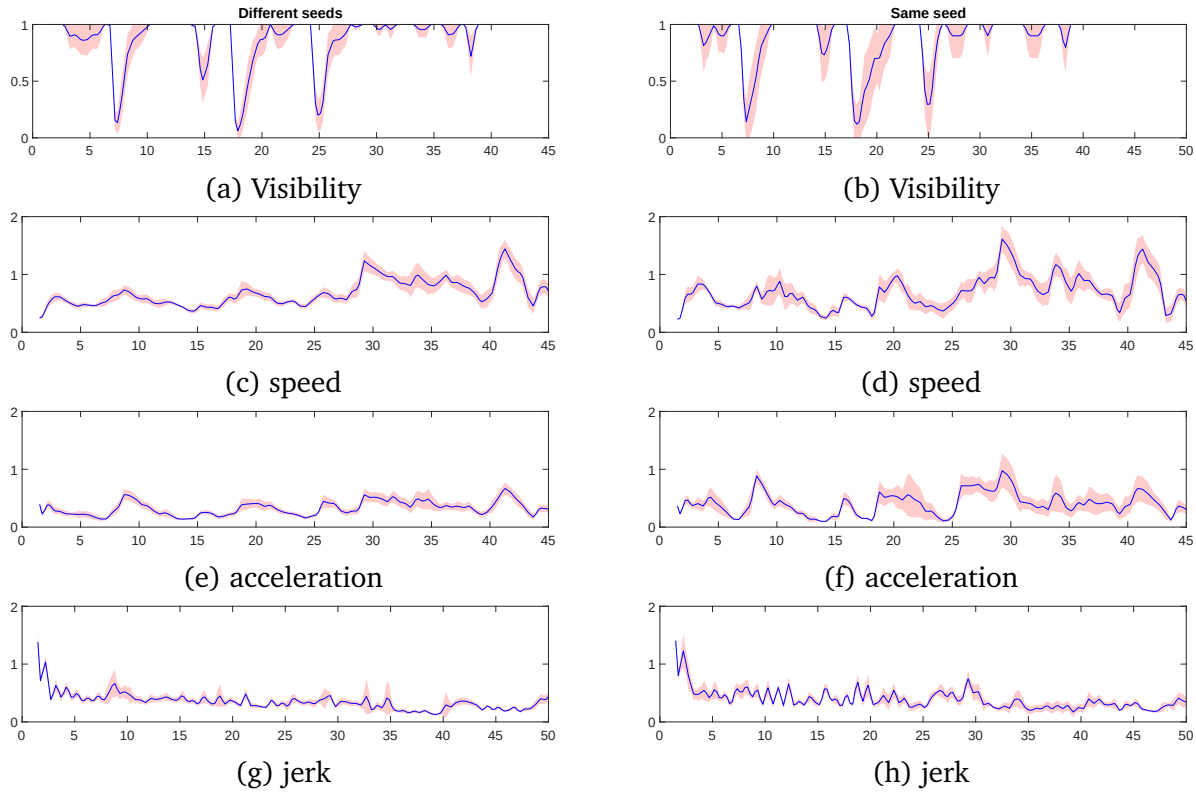


(b) Percentage of fails on each half-axis (-x, +x, -y, +y).



(c) Best animation score.

Figure 4.8 – Comparison of our system with an adaptive scale, or with a naïve scale, applied on the camera animation space.



(a) Visibility (b) Visibility

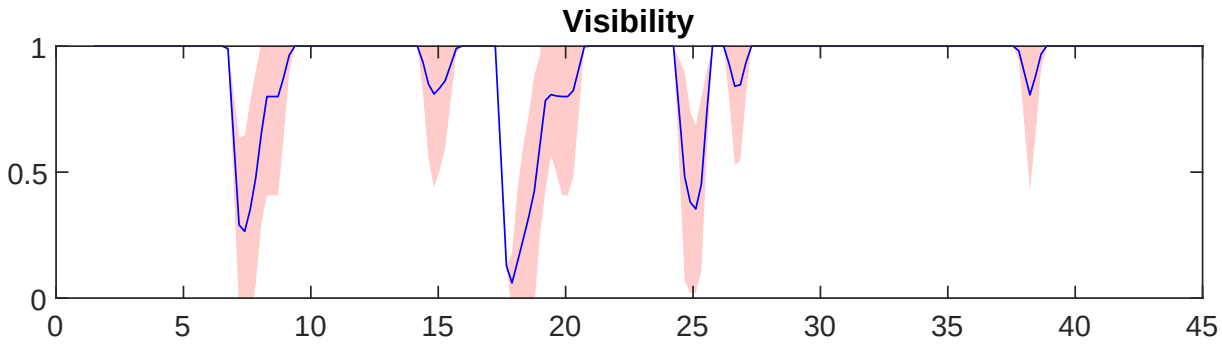
(c) speed (d) speed

(e) acceleration (f) acceleration

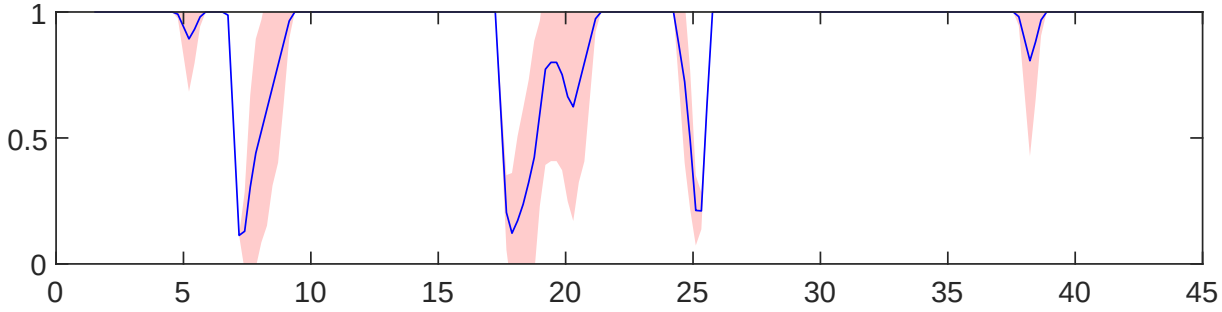
(g) jerk (h) jerk

Figure 4.9 – Results for multiple runs, each using a randomly generated camera animation space. This space is sampled with uniform distribution, with 2400 sample camera animations (Hermite curves). Each plot shows the mean value over time (blue), with a 95% confidence interval (red). Left: results for 22 runs using different seeds. Right: results for 10 runs using the same seed.

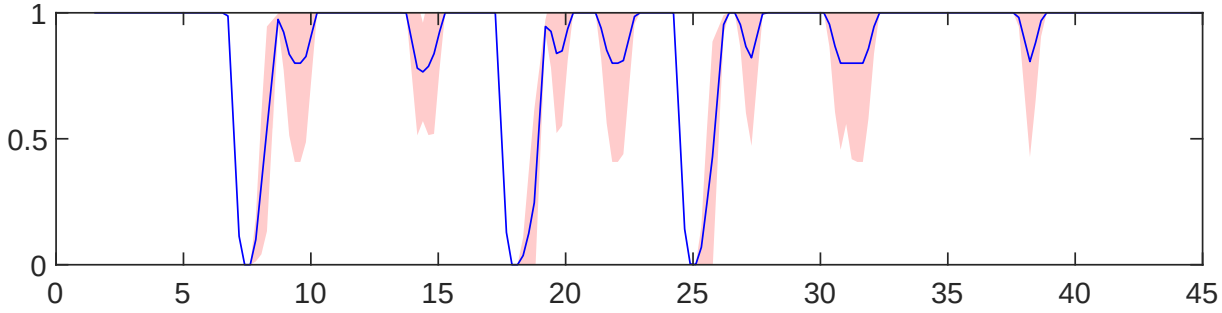
accompanying video). We validated our system by using (i) a **Toy example scene** where the target is travelling through a maze containing several tight corridors with sharp turns, an open area inside a building, and a ramp. We then performed the comparisons with the technique presented in the previous chapter, by using two static scenes and a dynamic scene, which the target goes through: (ii) a scene with a set of columns and a gate (**Columns+Gate**), (iii) a scene with set of small and large spheres (**Spheres**) and (iv) a fully dynamic scene with a set of randomly falling and rolling boxes, and a randomly sliding wall (**Dynamic**). To provide fair comparisons, in the dynamic scene, the random motions of boxes and of the wall are the same for both scenes. In addition, for all tests in a scene, we play a pre-recorded trajectory of the target avatar, but let the camera system run as if the avatar was interactively controlled by a user, to ensure motions are the same in all tests.



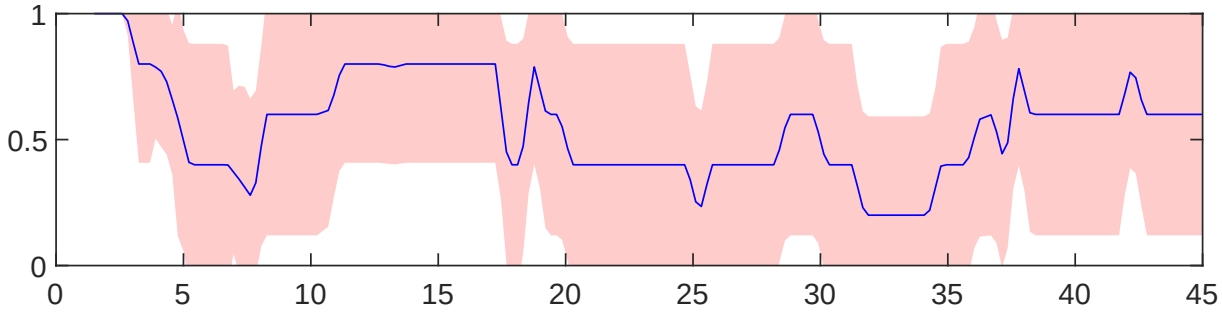
(a) 2400 camera animations



(b) 1600 camera animations



(c) 800 camera animations



(d) 100 camera animations

Figure 4.10 – Visibility when varying the number of sampled curves in our camera animation space.

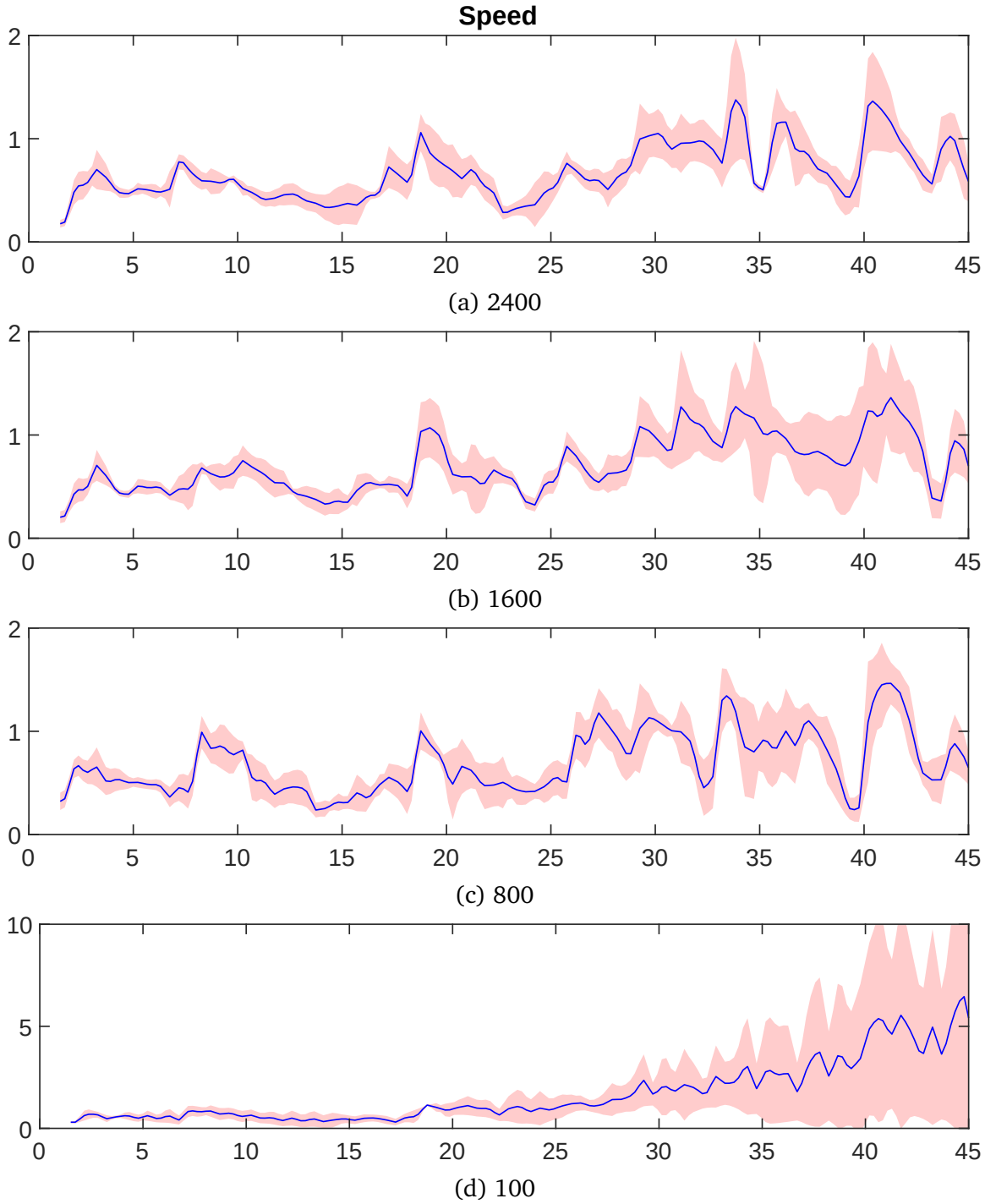


Figure 4.11 – Camera speed when varying the number of sampled curves in our camera animation space.

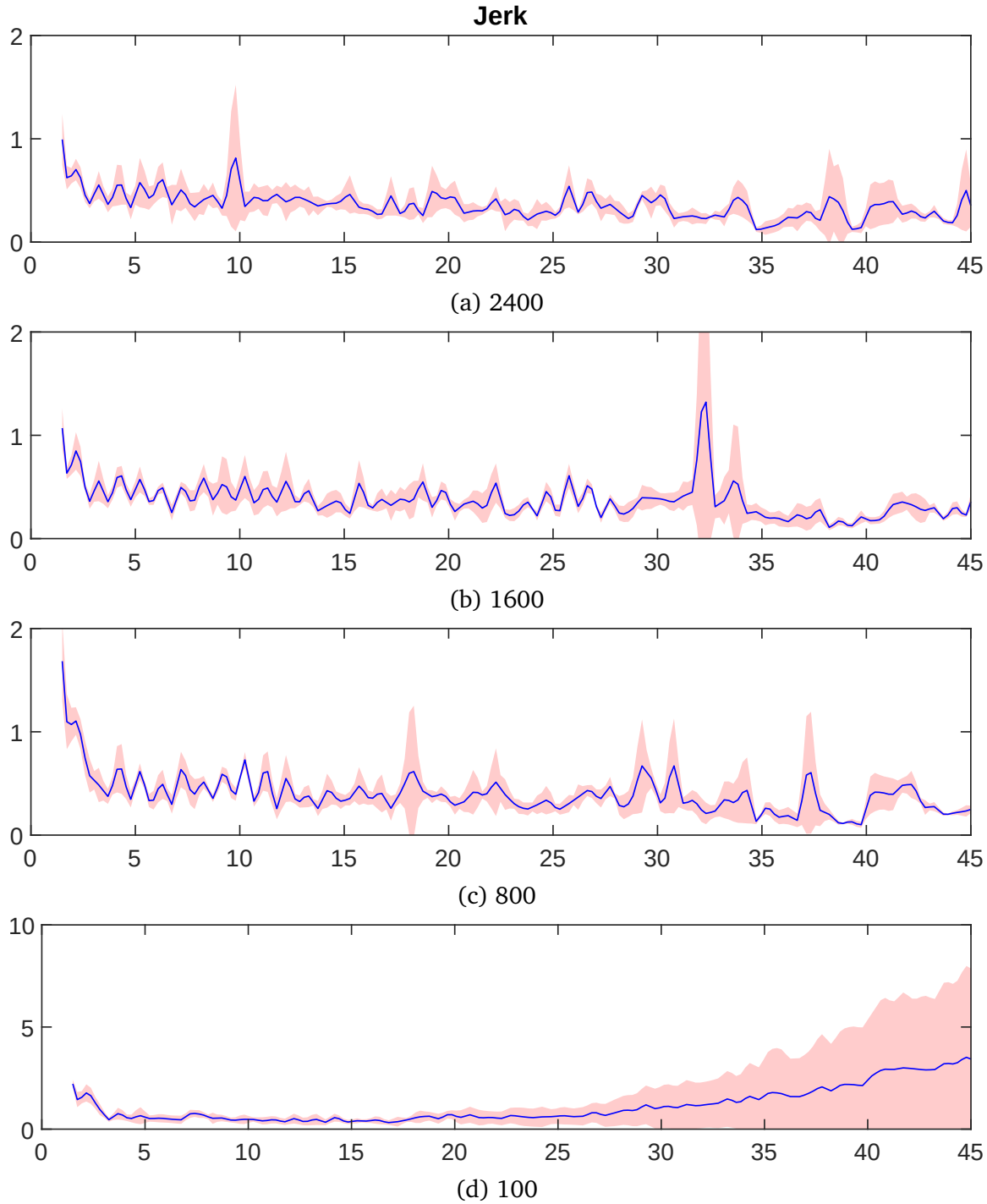
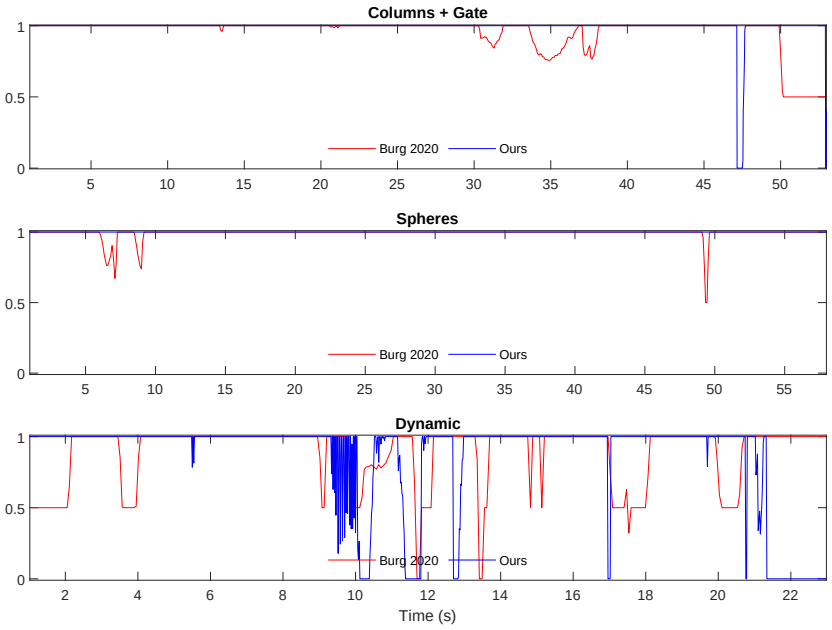
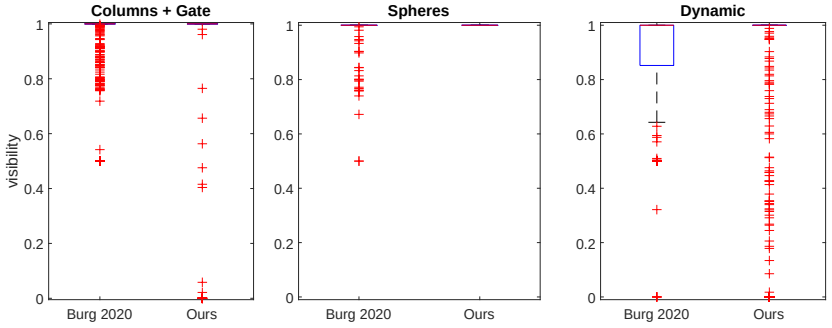


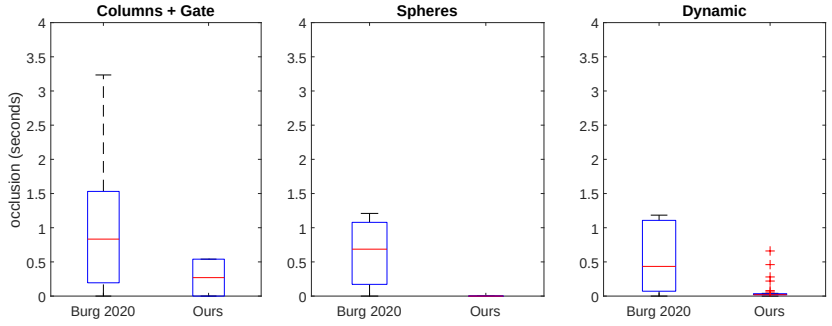
Figure 4.12 – Camera jerk when varying the number of sampled curves in our camera animation space.



(a) Visibility over time

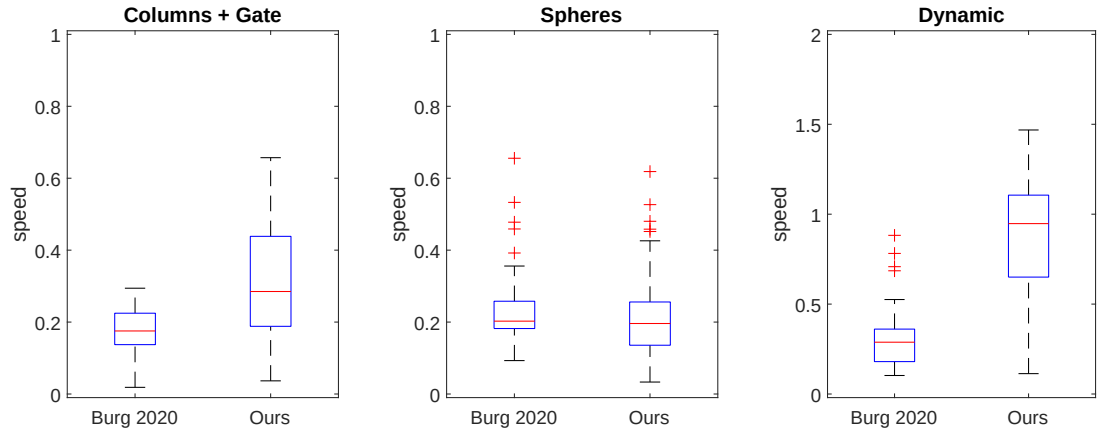


(b) Visibility ratio (distribution)

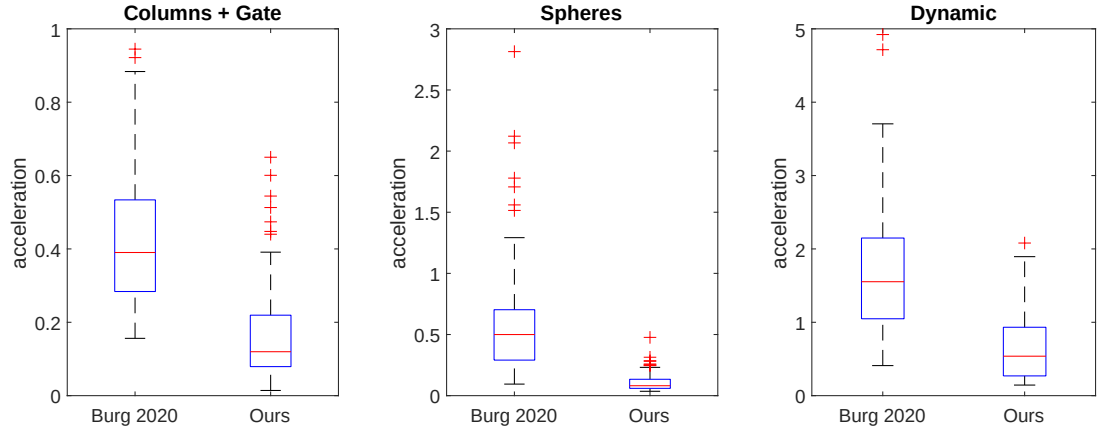


(c) Occlusion duration (distribution)

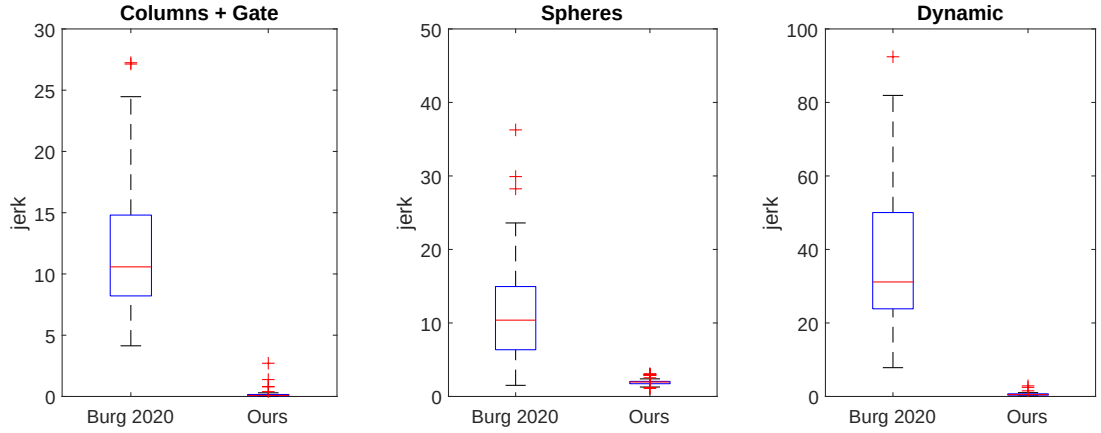
Figure 4.13 – Comparison between this approach and the chapter 3 [BLC20], regarding the target object's visibility (a)(b) and, when not fully visible, the duration of partial occlusion (c).



(a) Speed



(b) Acceleration



(c) Jerk

Figure 4.14 – Comparison between this approach the chapter 3 [BLC20], regarding the camera speed (a), acceleration (b) and jerk (c) distributions.

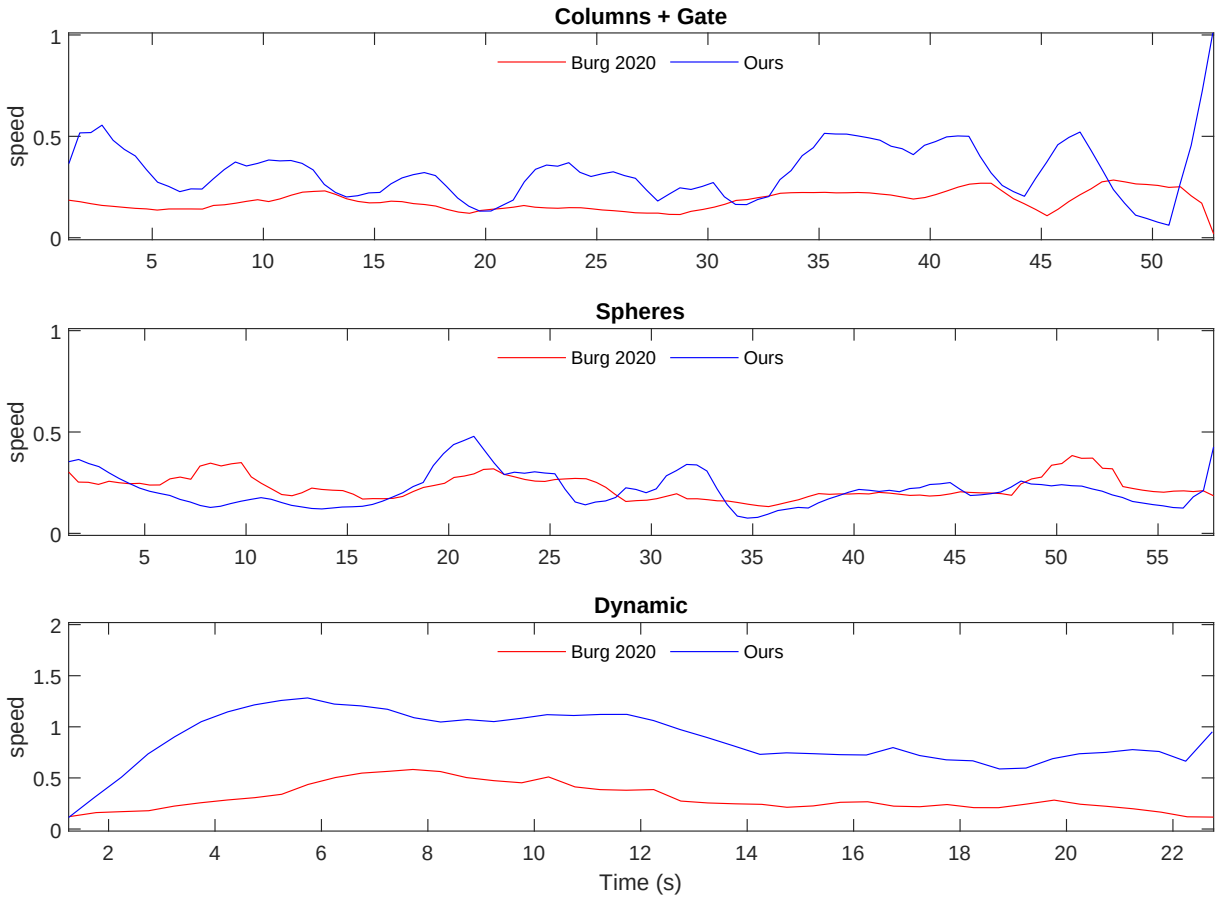


Figure 4.15 – Speed over time, for our camera system (blue) and the chapter 3 (red)

Impact of adaptive scale

We validate our adaptive scale (section 4.4.2) by comparing results obtained (i) when we compute and apply the *adaptive scale* on all 4 half-axes ($-x$,$+x$,$-y$,$+y$), and (ii) when we simply apply the same scale as for the z axis (which we will call the *naïve scale* technique). We ran our tests by using the toy example scene. For each technique, each time we evaluate a new set of camera animations, we output the new scale values and the ratio of fails on each half-axis. In addition, we output and plot the mean cost of the 5 best animations in this set. Results are presented in figure 4.8.

Figure 4.8a shows how much our mechanism tightens the animation space (compared to the naïve scaling technique) when the avatar is entering corridors, and grows back to the same scale when the avatar reaches less cluttered areas (*e.g.* in the open interior room, or the outdoor area). As expected, our mechanism allows to adapt the scale on half-spaces in a non-symmetrical way. As shown by figure 4.8b, with our adaptive

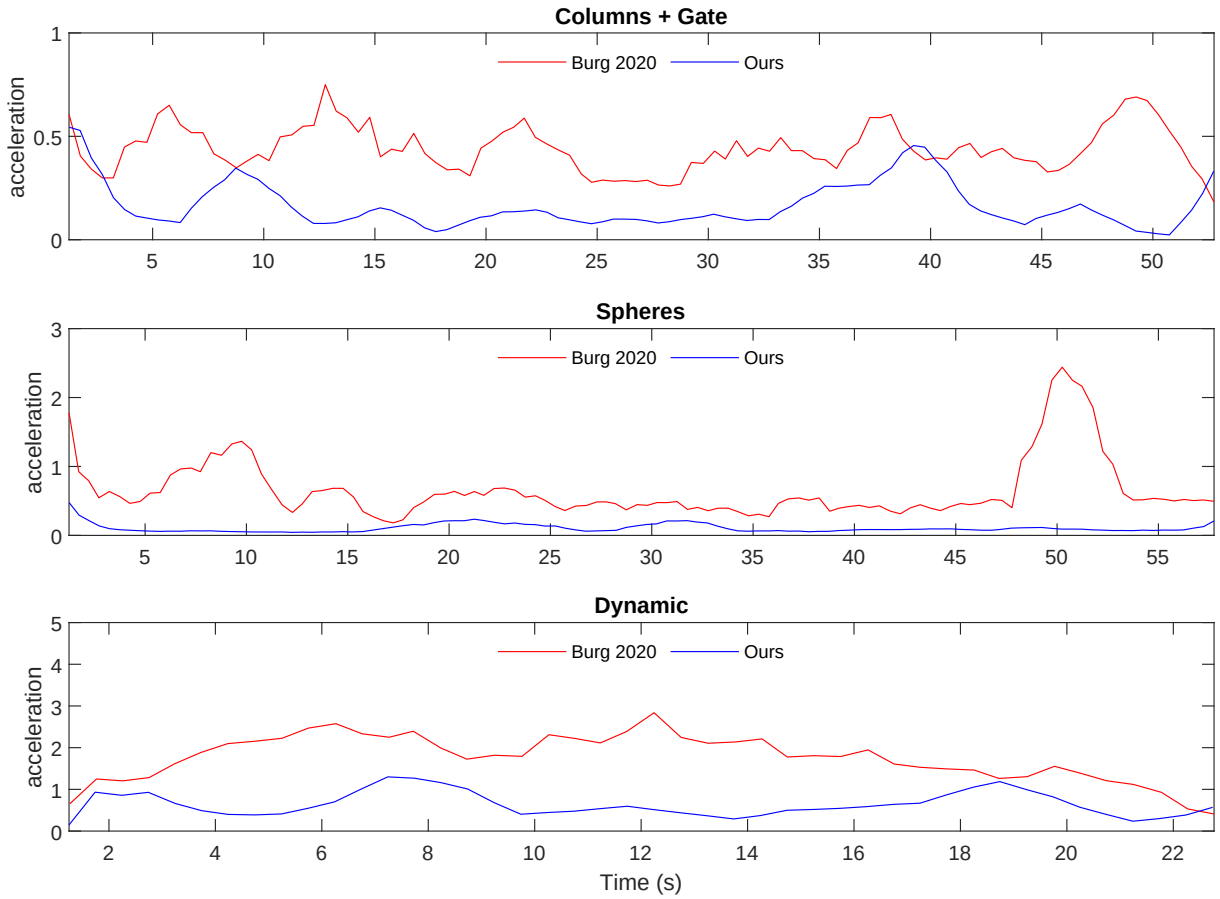


Figure 4.16 – Acceleration over time, for our camera system (blue) and the chapter 3 (red)

mechanism, the scaled animation space also exhibits fewer fails than using the naïve scale technique. We have shown by figure 4.8c, it allows finding animations with lower costs most of the time. One exception is between 40s and 50s, where the camera configuration isn't the same because the scale is different. In the naïve case the camera is high above the character while in the adaptive case, the camera is closer to the ground, thus the scores is not relevant in this case because the two configurations are too different to be compared.

In the next evaluations, we consider that the adaptive scale mechanism is always activated.

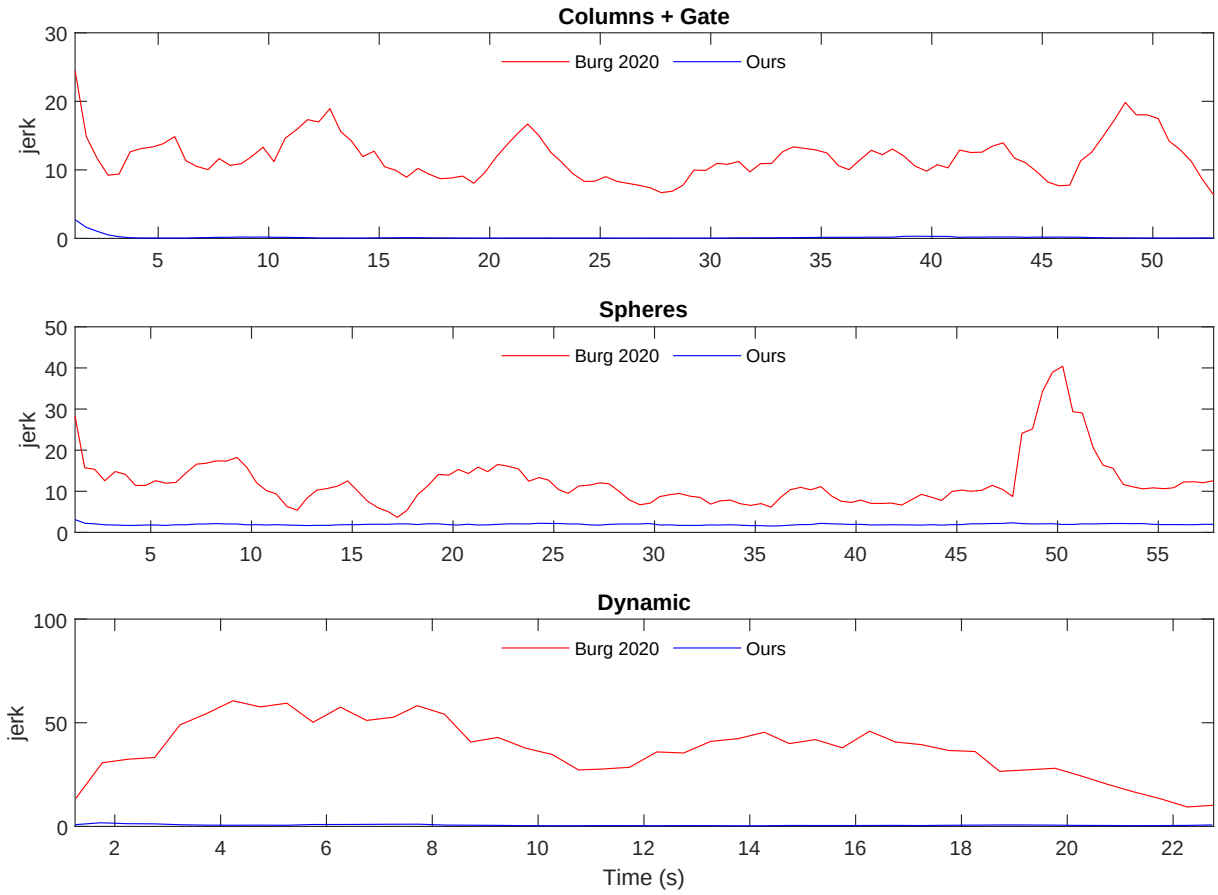


Figure 4.17 – Jerk over time, for our camera system (blue) and the chapter 3 (red)

Robustness

We study the robustness of our system regarding our randomly generated camera animation space.

In a first step, we evaluate how performances vary if we run our real-time system multiple times on the toy example scene. We also consider two cases: (i) using the same seed for every run (*i.e.* the same animation space is used), and (ii) using a new seed for every run (*i.e.* a new animation space is randomly sampled for each run). For each case, we sample a set of 2400 animations. Results are presented in figure 4.9. As illustrated, with as many sampled animations, all runs lead to very similar results both on the visibility enforcement and on the camera motion smoothness. Differences are mainly due to variations in the actual frame rate of the game engine, hence the rate at which the system takes new decisions.

In a second step, we evaluate how the size of the animation space (*i.e.* the number

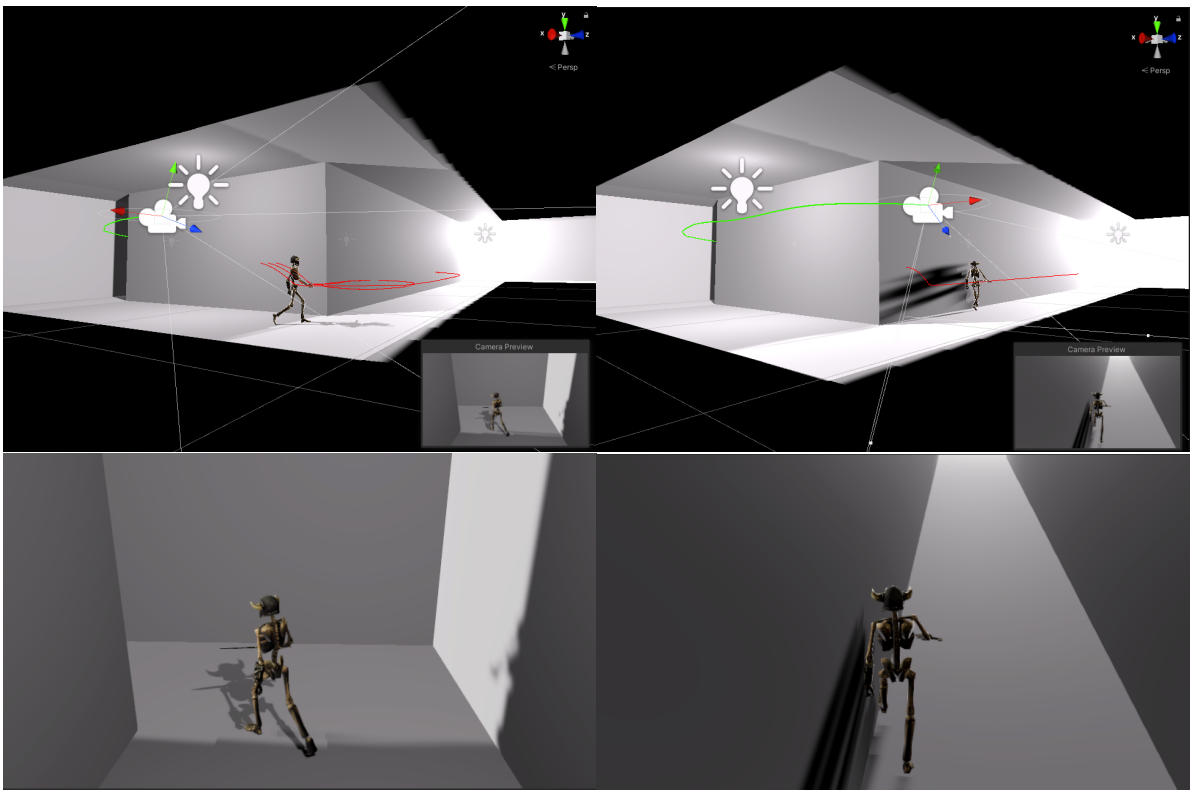


Figure 4.18 – Results on a narrow corridor: the target is the upper body of the character; The character performs a sharp turn and the camera manages to avoid colliding with the wall. The upper images are the scene views; the bottom images are the camera views; the left images are before the sharp turn; the right images are after the sharp turn.

of sampled animations) impacts performances. We ran our system with 4 different sizes: 2400, 1600, 800 or 100 animations. For each size, we performed 5 runs with random seed, and combined the results in figures 4.21, 4.22 and 4.23. Figures show that lowering the size (at least until 800 animations) still delivers good performances. Our camera system is able to find a series of camera animations maintaining enough visibility on the target object through smooth camera motions. As we expected, for 100 animations, our system's performances are poor: it becomes harder to find animations with sufficient visibility and ensuring smooth camera motions. Obviously as we lower the number of camera animations, the distribution of tangents becomes very sparse, hence breaking our assumption of a uniform sampling. If the sampled animation space does not cover enough the free space, it prevents the finding of qualitative animations.

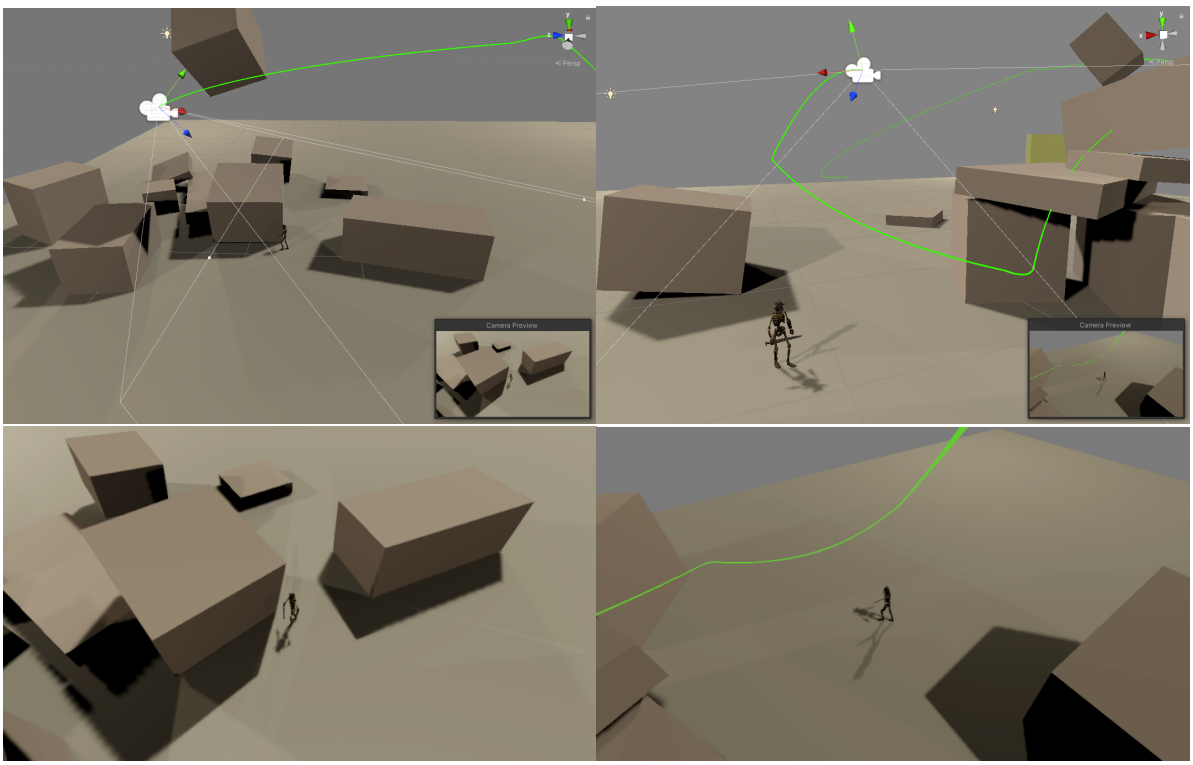


Figure 4.19 – Results on a scene with dynamic objects: the target is the upper body of the character; the character walks on a flat area with random falling objects thrown all around the scene. The camera manages to minimize occlusion and avoid most of the collisions (see figure 4.13a).

Comparison to previous work

We also compare our system mixing local and global planning approaches to local camera planning system presented in the previous chapter (*cf.* Chapter 3). We have run our proposed camera system and the local camera planning system in 3 different scenes: two static scenes (**Columns+Gate** and **Spheres**) and a fully dynamic scene (**Dynamic**). The **Columns + Gate** is the same as we used in the chapter 3 where the avatar is moving between some columns and go through a doorway. In the **Spheres** scene, the avatar is travelling a scene filled with a large set of spheres, which makes it moderately challenging for the camera systems. In the **Dynamic** scene, the avatar must go through a flat area, where a set of boxes are randomly flying, falling, rolling all over the place, and a wall is randomly sliding. This makes it challenging for camera systems to anticipate the scene dynamics and find occlusion-free and collision-free camera paths.

In our camera system, we sample 2400 animations in the animation space once at

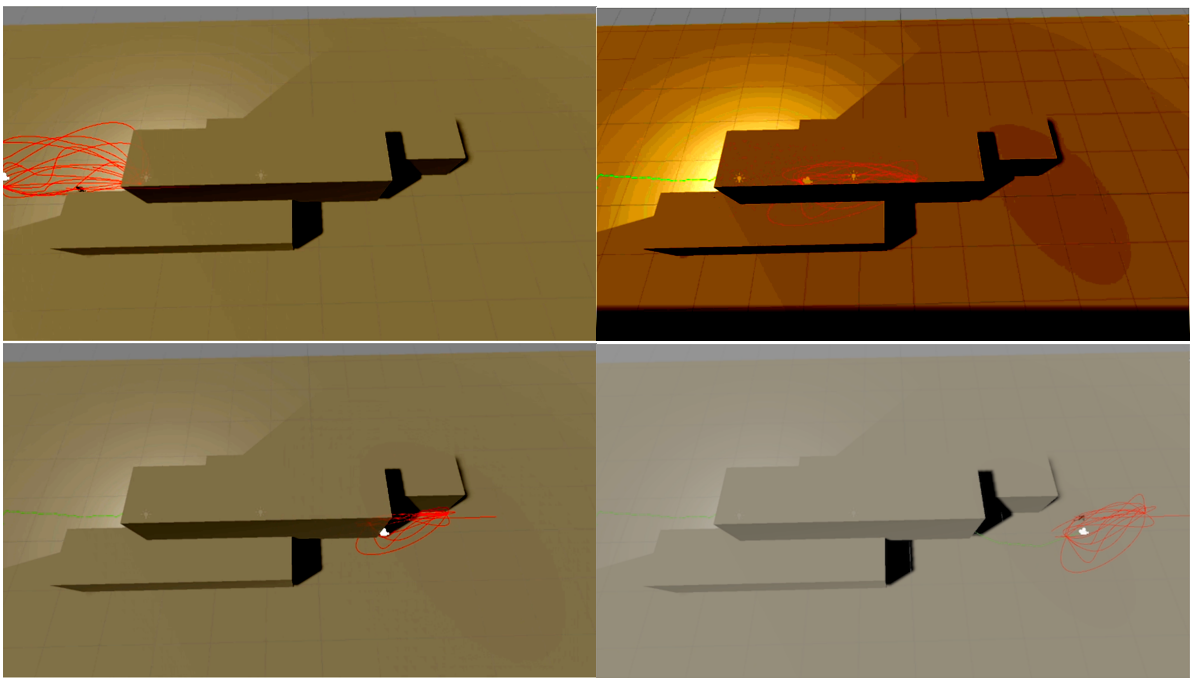
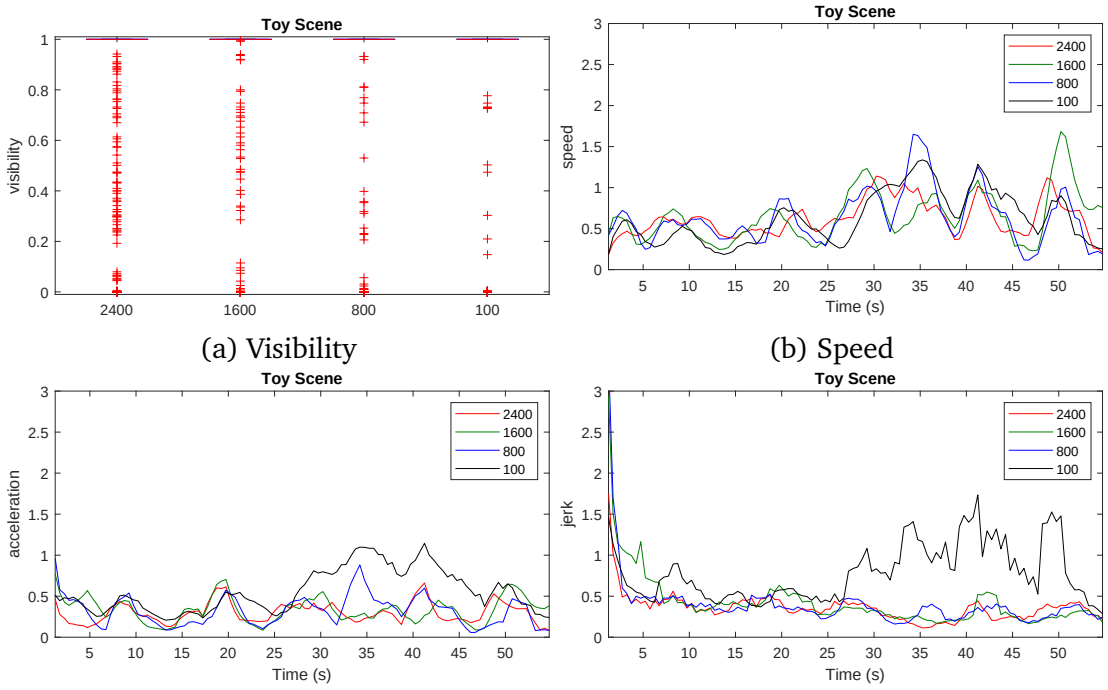


Figure 4.20 – Results on a scene with a corridor: we move the target through the corridor to illustrate how the animation space adapts to the environment. In red, we see 20 randomly picked trajectories from the animation space. In the first image (*i.e.* upper left) there is wall only on the right of the character, and we can already see that the right of the animation space has shrunken. In the second image (*i.e.* upper right), the character is surrounded by walls, and we can see that the animation space has shrunken in all directions. Finally, in the last two images as the character gets out of the corridor, the animation space widens to its original scales.

the beginning of the execution, the recomputation rate is set to 0.25s and the adaptive scaling is on. We present results of our tests in figures 4.13, 4.14, 4.15, 4.16, and 4.17.

We first compare camera systems along their ability to enforce visibility on the target object (figure 4.13). Our tests show that for moderately challenging scenes, both lead to relatively good results. Few occlusions occur. However, for a more challenging scene (**Dynamic**), our system outperforms the approach presented in the last chapter. Even if occlusions may occur more often, the degree of occlusion is lower (figure 4.13b). Moreover, for all 3 scenes, when partial occlusions occur, they are shorter when using our system (figure 4.13c). This is explained by the fact that when no local solution exist, our system can still find a locally occluded path respecting other constraints, and leading to a less occluded area. This demonstrates our system's ability to better anticipate occlusions especially in dynamic scenes.

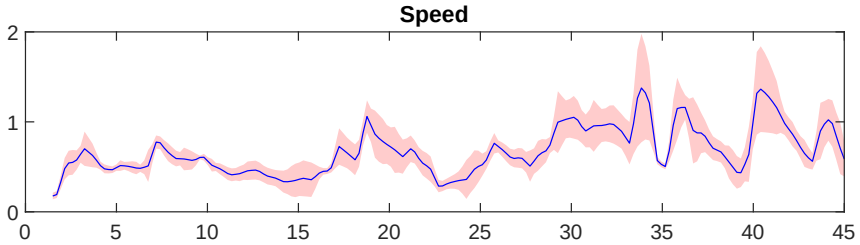


(a) Visibility (b) Speed

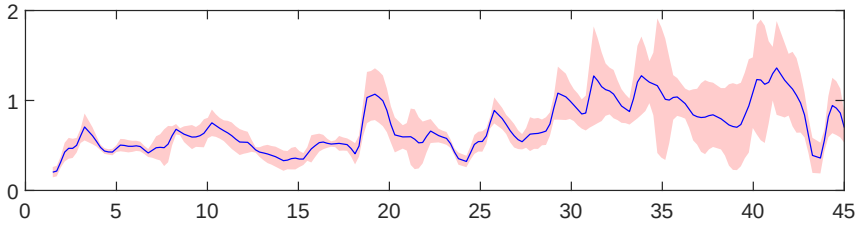
(c) Acceleration (d) Jerk

Figure 4.21 – Speed, acceleration and jerk when varying the number of sampled curves in our camera animation space.

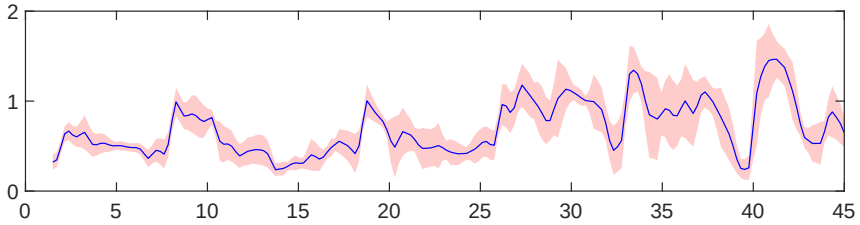
Second, we compare the smoothness of camera motions in both camera systems. Figure 4.14 presents the side-by-side distributions of speed, acceleration and jerk for each system. We also provide the speed, acceleration and jerk over time in figures 4.15, 4.16, and 4.17. One observation we make is that the previous approach system leads to lower camera speeds, as it restricts itself to simply following the avatar. In our camera system, the camera is allowed to move faster, to bypass the avatar when visibility or another constraint may be poorly satisfied. Yet, our system provides smoother motions (*i.e.* less jerk). One explanation is that local systems often need to steer the camera from local minima (*e.g.* low visibility areas). A side effect is that it may lead, for successive iterations, to an indecision on which direction the camera should take to reach better visibility. In turn, this leads to frequent changes in camera acceleration (hence higher jerk). Conversely, our system has a more global knowledge on the scene, allowing to more easily find a better path, which avoids sacrificing the smoothness of camera motions.



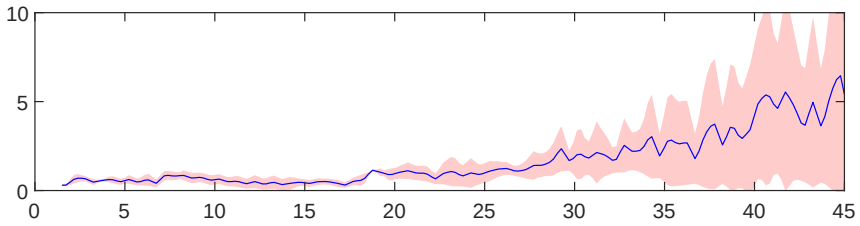
(a) 2400 camera animations



(b) 1600 camera animations



(c) 800 camera animations



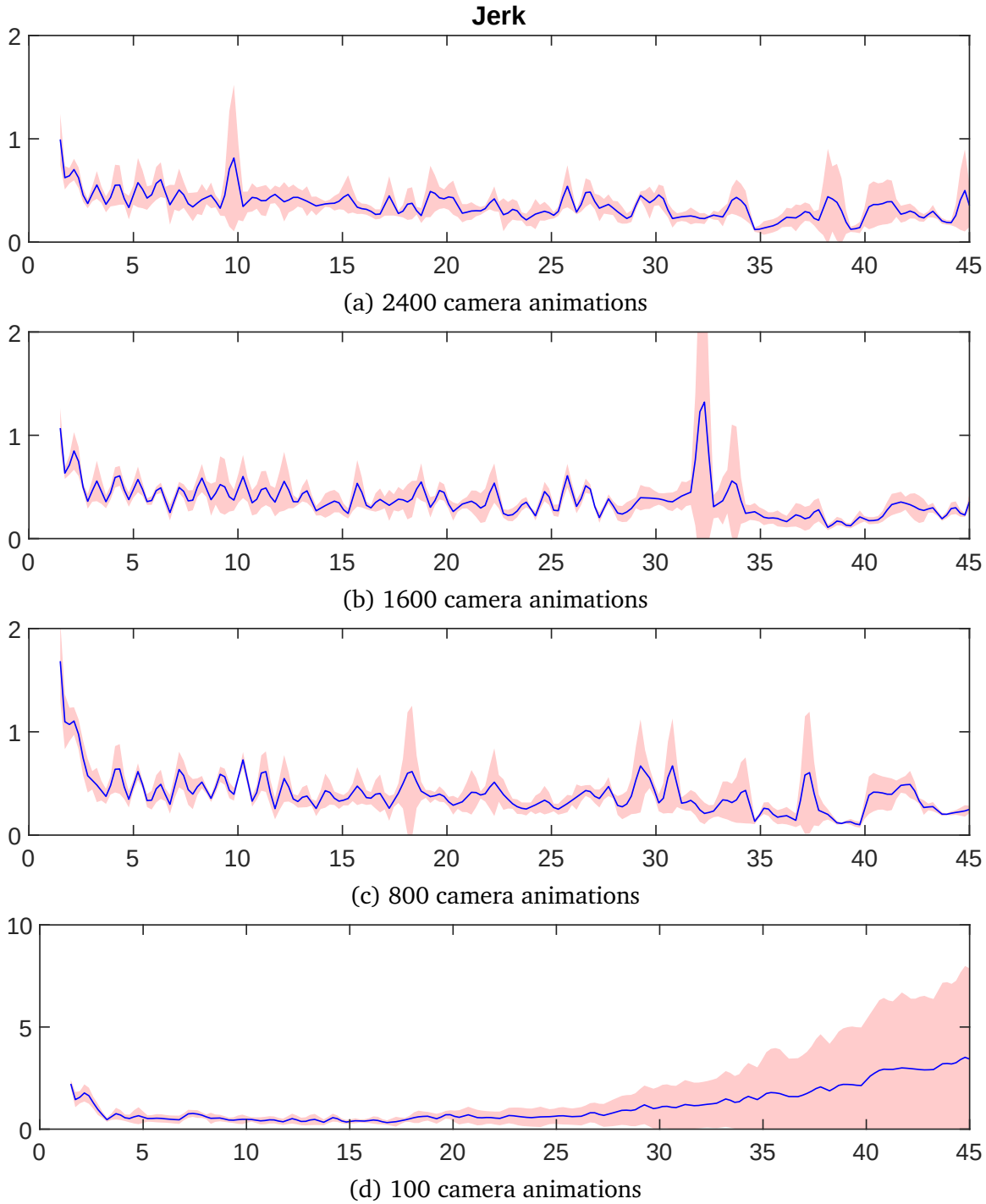
(d) 100 camera animations

Figure 4.22 – Camera speed when varying the number of sampled curves in our camera animation space.

4.5.3 Discussion and limitations

We report in this section some error cases and difficulties that the system can have on specific situations.

First, as the system has a lot of constraints to satisfy, the decisions made on the weights is critical to ensure that the camera has the desired behavior. With wrong weight design the camera will hardly manage to converge towards a stable trajectory or will go through walls (Figure 4.24a).



(a) 2400 camera animations

(b) 1600 camera animations

(c) 800 camera animations

(d) 100 camera animations

Figure 4.23 – Camera jerk when varying the number of sampled curves in our camera animation space.

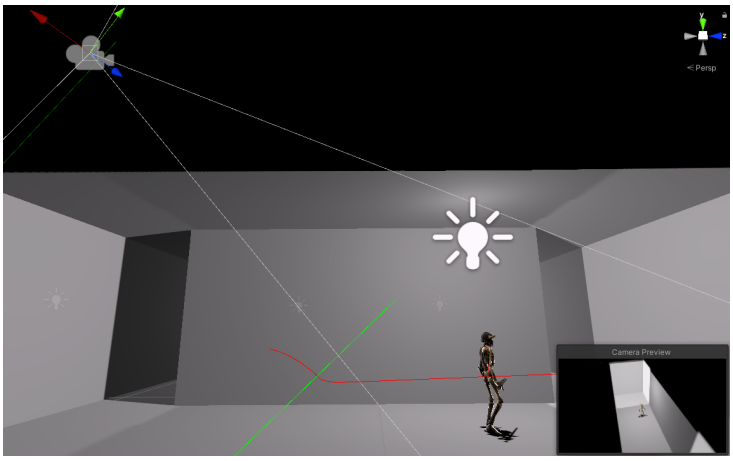
Second, the animation space sampling does not allow the system to find collision-free solution most of the time on really narrow environments (see Figure 4.24a). Moreover, for performance's sake we are not able to significantly increase the number of samples from the animation space. Besides, if a collision free trajectory is found, the quality of the shots will be poor (*i.e.* not collision free or with an incorrect camera angle see Figure 4.24b and 4.24c).

4.6 Conclusion

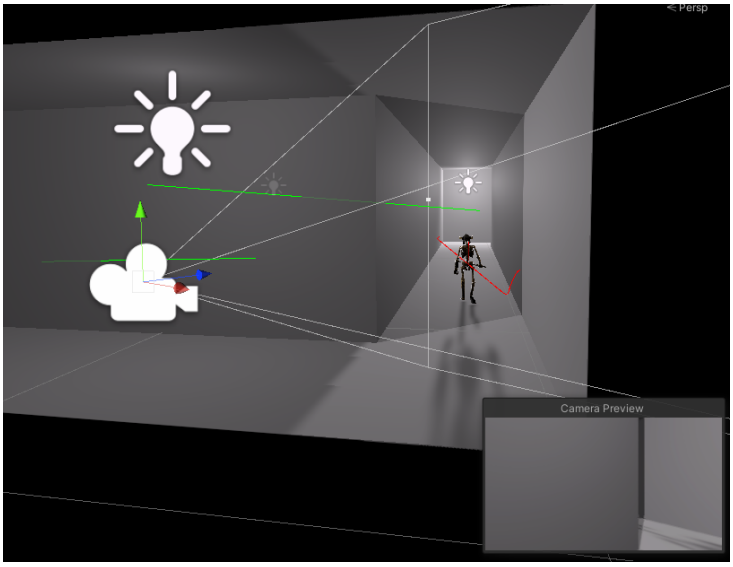
In this chapter we proposed an efficient camera tracking technique adapted to dynamic 3D environments that does not require heavy roadmap pre-computations. We defined a parametric camera animation space, which adapts to the 3D environment. We sample the animation space using quadratic Hermite curves that we evaluate in real-time with RTX technology and GPU computation.

Despite the ability of our system to evaluate thousands of trajectories each frame, strongly cluttered environments remain challenging. As smoothness is enforced, visibility may be lost in specific cases. Designing a technique that could properly balance between the different properties in order to handle specific cases, still needs to be addressed.

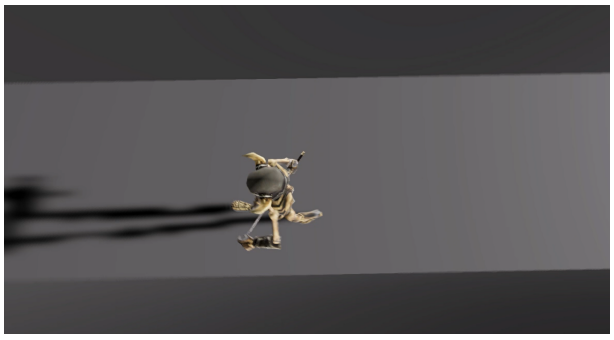
A future work and straightforward work could consist in biasing the sampling in the animation space in order to adapt the space to typical local topologies of the 3D environment to sample less trajectories that are more relevant.



(a) Camera collision on a narrow environment



(b) Occluded target



(c) Result of a compromise between aesthetics and collision due to few collision-free camera tracks available.

Figure 4.24 – Illustration of some limitations.

CHAPTER 5

CONCLUSION

After exposing the current state-of-the-art in computational virtual cinematography and identifying the gaps and challenges of the field, we proposed two approaches to solve the problem of real-time target tracking in dynamic virtual environment using rendering techniques. Both contributions were accepted and presented respectively in Eurographics 2020 and Graphics Interface 2021.

Our first contribution is a local approach to the target tracking problem (*cf.* section 2.4.2). We use the Toric space representation to solve the framing constraint for two targets. Then, we rely on rasterization to project visibility on the Toric surface to compute a map of the local surrounding around the camera. We propose an anticipation model and accordingly a motion model to avoid dynamic occluders. Finally, to account for style we propose to enforce specific view angles by adding a mask to the Toric space. The limit of this contribution is inherent to its local nature (*i.e.* even with our anticipation model, as the searching technique has very little knowledge on the environment, it can easily stay stuck in local minima).

To overcome this problem, we propose a second contribution, that uses a hybrid approach (*cf.* section 2.4.3). We focus on one character only. By anticipating the character motion we are able to define an ideal target position for the camera in the future. Then we search a good trajectory from the current camera position to the target camera position. For this we introduce the camera animation space, a space that defines all camera animations between two points. We sample the camera animation space with quadratic Hermite trajectory model. Then, we rely on RTX technology to evaluate our criteria (*i.e.* visibility, collision, view angle, motion) that we use to rank the trajectories and select the best one to for the camera to play. In order to avoid dynamic elements in the environment, we perform this process using MPC with a short recomputing frequency (0.25s). Despite the fact that the tracking gives good results in open areas, in narrow areas, the optimization process failed in finding collision-free trajectories. The reason behind this is pertained to limitations in the sampling of the camera animation space.

5.1 Perspectives

Toward an environment aware distribution in the camera animation space. After defining the animation space, we used a uniform sampling of the animation space and noticed that this still presented limitations for cluttered environment. To go further we could bias the sampling depending on the local 3D environments. We make two hypotheses: (i) by analyzing the environment using the depth information around the $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ (*e.g.* 360 degree maps) we could bias the probability field for the tangent distribution; (ii) using data driven approach we could learn the trajectory distribution based on the depth information.

Defining a quality metric for camera motions. We used a straightforward function to define each camera track quality, but the quality is still not well-defined in the community (*cf.* section 2.4.5) and it is necessary to define the meaning of "quality" when addressing the problem of automated camera trajectory generation. Similarly to editing, (see Lino *et al.* [Lin+14]) we should be able to define a quality metric for camera trajectories. There is a long history of cinema and a good starting point would be to analyze and create a latent space of the existing camera trajectories. This has already by done by others on different fields. For example Wei *et al.* [Wei+18] proposed a deep learning method to find well composed views inside 2D images. The system they propose has an evaluation network that is able to judge if a view is better than another. In other words, a comparison loss function has been designed to compare two views. We could rely on a similar idea by transposing the approach to camera trajectories. However, trajectories are more complex to judge compares to single images, because of the importance of the context and narrative intention.

Towards data-driven camera trajectories. Exploiting data from existing films rises a number of questions:

1. **On what basis do we judge a real camera trajectory ?** We could use a movie database and use the films rating system to favor some motions over others, but usually the evaluation includes other parameters such as the scenario and production quality that are irrelevant in our case. Another solution would be to extract camera trajectories from movies (*e.g.* structure from motion, dense or sparse SLAM techniques). The problem with this solution is that such extraction tends

to produce noisy results and are sensitive to cinematic features such as depth of field, blur, or low light conditions. Moreover, we cannot extract with precision the relation between the targets and the camera and determine the amount of motion of both.

2. **Are real camera trajectories enough to define what is a good virtual camera trajectory ?** If we can extract trajectories from real movies, we will be able to generate trajectories similar to the ones we extracted, but real cameras have physical constraints that virtual camera are freed from. Meaning that, among all possible trajectories for virtual cameras, a dataset composed by camera trajectories extracted from films only, might not be enough to cover all possible good trajectories. We could increase the dataset with virtual camera trajectories, but we still do not know how to generate "good" trajectories. This therefore, raises the question, whether virtual cinematography itself has developed, or needs to develop its own style, and dedicated quality metric.
3. **Whet level of information is necessary to define a good camera trajectory ?** We can probably define a pleasant camera trajectory, however, for a camera trajectory to be actually "good" it has to match the context of the story, the personality of the characters, and the mood of the scene, three subjective criteria, among many others.

Defining the camera trajectory styles. While searching for these answers we are often faced with the problem of defining what is the camera trajectory style. The notion of style is subtle. In literature for example, the writing style is defined by the choice of words, sentence structures and paragraph structures. In other words, writers use their tools (*e.g.* words, grammar, literary device) and organize them at multiple levels (*i.e.* sentence, paragraph) according to the story context in order to define their style. As a parallel, filmmakers also have low level tools (*e.g.* camera angle, camera distance, image composition, editing, etc...) that they use and organize at multiple levels (*i.e.* shot, scene) according to the story context.

At this point we can assume that a storytelling style can be defined by the balance and frequency of the use of elementary bricks (*i.e.* an association of the camera parameters) to convey a message.

If we dive deeper in this idea, we could imagine defining bricks of the camera tracking style as an association of low level parameters' evolution on a short period of time

(*e.g.* camera angle, distance, composition, camera speed, background motion, etc...) and its overall style as a balance and frequency of the use of each brick.

Learning good camera trajectory from user preferences. Finally, a camera trajectory quality is subjective to the viewer perception and the community of virtual cinematography has not relied enough on user studies (*cf.* 2.4.5). Moreover, a lot of data on user preferences related to the question of quality for camera trajectories can be exploited in a learning process (see [Bon+20a]) and could lead us to the understanding of either the quality or the style of a camera trajectory. Once more, data driven approaches could be a good way to process the user studies data and help us determine if a trajectory is qualitative and what style does it belong to.

BIBLIOGRAPHY

[AA10] Ferran Argelaguet and Carlos Andujar, « Automatic speed graph generation for predefined camera paths », *in*: *International Symposium on Smart Graphics*, Springer, 2010, pp. 115–126 (cit. on pp. 42, 51).

[Abd+11] Rafid Abdullah et al., « Advanced composition in virtual camera control », *in*: *International Symposium on Smart Graphics*, Springer, 2011, pp. 13–24 (cit. on pp. 33, 50).

[Ajm+12] Muhammad Ajmal et al., « Video summarization: techniques and classification », *in*: *International Conference on Computer Vision and Graphics*, Springer, 2012, pp. 1–13.

[AM16] Ferran Argelaguet and Morgant Maignant, « GiAnt: stereoscopic-compliant multi-scale navigation in VEs », *in*: *Proceedings of the 22nd acm conference on virtual reality software and technology*, 2016, pp. 269–277.

[Ari91] D. Arijon, *Grammar of the Film Language*, Silman-James Press, 1991, ISBN: 9781879505070, URL: https://books.google.fr/books?id=6bQlAQAAIAAJ (cit. on p. 31).

[Ass+08] Jackie Assa et al., « Motion overview of human actions », *in*: *ACM Transactions on Graphics (TOG)* 27.5 (2008), pp. 1–10 (cit. on pp. 36, 43).

[AWC10] Jackie Assa, Lior Wolf, and Daniel Cohen-Or, « The virtual director: a correlation-based online viewing of human motion », *in*: *Computer Graphics Forum*, vol. 29, 2, Wiley Online Library, 2010, pp. 595–604 (cit. on pp. 36, 51).

[BBS20] Arthur Bucker, Rogerio Bonatti, and Sebastian Scherer, « Do You See What I See? Coordinating Multiple Aerial Cameras for Robot Cinematography », *in*: *arXiv preprint arXiv:2011.05437* (2020) (cit. on pp. 44, 51).

[BCM21] Alexandre Bruckert, Marc Christie, and Olivier Le Meur, « Where to look at the movies: Analyzing visual attention to understand movie editing », *in*: *arXiv preprint arXiv:2102.13378* (2021) (cit. on p. 51).

[BJ09] Paolo Burelli and Arnav Jhala, « Dynamic artificial potential fields for autonomous camera control », *in*: *Artificial Intelligence and Interactive Digital Entertainment* 1 (2009) (cit. on p. 41).

[BK00] Robert Bohlin and Lydia E Kavraki, « Path planning using lazy PRM », *in*: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1, IEEE, 2000, pp. 521–528 (cit. on p. 42).

[BLC20] Ludovic Burg, Christophe Lino, and Marc Christie, « Real-time Anticipation of Occlusions for Automated Camera Control in Toric Space », *in*: *Computer Graphics Forum*, vol. 39, 2, Wiley Online Library, 2020, pp. 523–533 (cit. on pp. 11, 20, 103, 104).

[BLC21] Ludovic Burg, Christophe Lino, and Marc Christie, « Real-Time Cinematic Tracking of Targets in Dynamic Environments », *in*: (2021) (cit. on pp. 12, 21).

[Bli88] Jim Blinn, « Where am I? What am I looking at?(cinematography) », *in*: *IEEE Computer Graphics and Applications* 8.4 (1988), pp. 76–81 (cit. on pp. 23, 26, 32, 33).

[BMH98] Dirk Bartz, Michael Meißner, and Tobias Hüttner, « Extending Graphics Hardware For Occlusion Queries In OpenGL. », *in*: *Workshop on Graphics Hardware*, 1998.

[Bon+18] Rogerio Bonatti et al., « Autonomous drone cinematographer: Using artistic principles to create smooth, safe, occlusion-free trajectories for aerial filming », *in*: *International Symposium on Experimental Robotics*, Springer, 2018, pp. 119–129 (cit. on pp. 44, 48, 50).

[Bon+19] Rogerio Bonatti et al., « Towards a robust aerial cinematography platform: Localizing and tracking moving targets in unstructured environments », *in*: *arXiv preprint arXiv:1904.02319* (2019) (cit. on pp. 44, 48).

[Bon+20a] Rogerio Bonatti et al., « Autonomous aerial cinematography in unstructured environments with learned artistic decision-making », *in*: *Journal of Field Robotics* 37.4 (2020), pp. 606–641 (cit. on pp. 48, 51, 120).

[Bon+20b] Rogerio Bonatti et al., « Batteries, camera, action! Learning a semantic control space for expressive robot cinematography », *in*: *arXiv preprint arXiv:2011.10118* (2020) (cit. on p. 52).

[Bou14] Tamy Boubekeur, « ShellCam: Interactive geometry-aware virtual camera control », *in*: *2014 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2014, pp. 4003–4007 (cit. on pp. 29, 30).

[Bur+02] Nicholas Burtnyk et al., « Stylecam: interactive stylized 3d navigation using integrated spatial & temporal controls », *in*: *Proceedings of the 15th annual ACM symposium on User interface software and technology*, 2002, pp. 101–110.

[BY10a] Paolo Burelli and Georgios N Yannakakis, « Combining local and global optimisation for virtual camera control », *in*: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, IEEE, 2010, pp. 403–410 (cit. on p. 42).

[BY10b] Paolo Burelli and Georgios N Yannakakis, « Global search for occlusion minimisation in virtual camera control », *in*: *IEEE Congress on Evolutionary Computation*, IEEE, 2010, pp. 1–8 (cit. on pp. 42, 49).

[Cao+17] Zhe Cao et al., « Realtime multi-person 2d pose estimation using part affinity fields », *in*: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7291–7299 (cit. on p. 47).

[Che+09] Billy Chen et al., « Integrated videos and maps for driving directions », *in*: *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, 2009, pp. 223–232.

[CN05] Marc Christie and Jean-Marie Normand, « A semantic space partitioning approach to virtual camera composition », *in*: *Computer Graphics Forum*, vol. 24, 3, Blackwell Publishing, Inc Oxford, UK and Boston, USA, 2005, pp. 247–256 (cit. on p. 39).

[CNO12] Marc Christie, Jean-Marie Normand, and Patrick Olivier, « Occlusion-free camera control for multiple targets », *in*: *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2012 (cit. on pp. 41, 49, 59, 63, 83).

[Coh+03] Daniel Cohen-Or et al., « A survey of visibility for walkthrough applications », *in*: *IEEE Transactions on Visualization and Computer Graphics* 9.*3* (2003), pp. 412–431 (cit. on p. 58).

[CON08] Marc Christie, Patrick Olivier, and Jean-Marie Normand, « Camera control in computer graphics », *in*: *Computer Graphics Forum*, vol. 27, 8, Wiley Online Library, 2008, pp. 2197–2218 (cit. on pp. 23, 27, 31, 49, 59).

[Eis+16] Elmar Eisemann et al., *Real-time shadows*, AK Peters/CRC Press, 2016 (cit. on p. 58).

[ER07] David Elson and Mark Riedl, « A lightweight intelligent virtual cinematography system for machinima production », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 3, 1, 2007, pp. 8–13 (cit. on p. 25).

[ETB15] Zaynab Elkhattabi, Youness Tabii, and Abdelhamid Benkaddour, « Video summarization: techniques and applications », *in*: *International Journal of Computer and Information Engineering* 9.*4* (2015), pp. 928–933.

[Eva20] Contis Eva, *Film Techniques for Students and Beginner Filmmakers*, 2020, URL: https://www.careersinfilm.com/film-techniques/ (cit. on pp. 5, 15).

[Evi+20] Inan Evin et al., « 3pp-r: Enabling natural movement in 3rd person virtual reality », *in*: *Proceedings of the annual symposium on computer-human interaction in play*, 2020, pp. 438–449.

[FB13] Gunnar Flötteröd and Michel Bierlaire, « Metropolis–Hastings sampling of paths », *in*: *Transportation Research Part B: Methodological* 48 (2013), pp. 53–66 (cit. on p. 48).

[Fee04] Catherine Feeny, *VFXPro - The Daily Visual Effects Resource*, 2004, URL: https://web.archive.org/web/20040318114836/http://www.uemedia.net/CPC/vfxpro/article_7062.shtml (cit. on pp. 6, 16).

[Gal+13] Quentin Galvane et al., « Steering behaviors for autonomous cameras », *in*: *Proceedings of Motion on Games*, 2013, pp. 93–102 (cit. on pp. 37, 41, 50, 67).

[Gal+15a] Quentin Galvane et al., « Camera-on-rails: automated computation of constrained camera paths », *in*: *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, 2015, pp. 151–157 (cit. on p. 50).

[Gal+15b] Quentin Galvane et al., « Continuity editing for 3D animation », *in*: *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Gal+18a] Quentin Galvane et al., « Directing cinematographic drones », *in*: *ACM Transactions on Graphics (TOG)* 37.*3* (2018), pp. 1–18 (cit. on pp. 44, 52).

[Gal+18b] Quentin Galvane et al., « Directing the Photography: Combining Cinematic Rules, Indirect Light Controls and Lighting-by-Example », *in*: *Computer Graphics Forum*, vol. 37, 7, Wiley Online Library, 2018, pp. 45–53 (cit. on p. 25).

[Geb+16] Christoph Gebhardt et al., « Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals », *in*: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 2508–2519 (cit. on pp. 40, 45, 50).

[Ger09] Roland Geraerts, « Camera planning in virtual environments using the corridor map method », *in*: *International Workshop on Motion in Games*, Springer, 2009, pp. 194–206 (cit. on p. 39).

[GH21] Christoph Gebhardt and Otmar Hilliges, « Optimization-based User Support for Cinematographic Quadrotor Camera Target Framing », *in*: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–13.

[Gsc+19] Mirko Gschwindt et al., « Can a robot become a movie director? Learning artistic principles for aerial cinematography », *in*: *arXiv preprint arXiv:1904.02579* (2019) (cit. on pp. 47, 48).

[GSH18] Christoph Gebhardt, Stefan Stevšić, and Otmar Hilliges, « Optimizing for aesthetically pleasing quadrotor camera motion », *in*: *ACM Transactions on Graphics (TOG)* 37.*4* (2018), pp. 1–11 (cit. on p. 40).

[GW92] Michael Gleicher and Andrew Witkin, « Through-the-lens camera control », *in*: *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, 1992, pp. 331–340 (cit. on pp. 26, 32).

[HA+92] Yong Koo Hwang, Narendra Ahuja, et al., « A potential field approach to path planning. », *in*: *IEEE transactions on robotics and automation* 8.*1* (1992), pp. 23–32 (cit. on pp. 28, 38).

[Hac+08] Martin Hachet et al., « Navidget for easy 3d camera positioning from 2d inputs », *in*: *2008 IEEE Symposium on 3D User Interfaces*, Ieee, 2008, pp. 83–89 (cit. on p. 30).

[Han+22] R Hanocka et al., « Learning Camera Control in Dynamic Scenes from Limited Demonstrations », *in*: *Computer Graphics Forum*, Wiley Online Library, 2022 (cit. on p. 46).

[HCS96] Li-wei He, Michael F Cohen, and David H Salesin, « The virtual cinematographer: a paradigm for automatic real-time camera control and directing », *in*: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 217–224.

[HH97] Paul S Heckbert and Michael Herf, *Simulating soft shadows with graphics hardware*, tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1997 (cit. on p. 58).

[HHS01] Nicolas Halper, Ralf Helbing, and Thomas Strothotte, « A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence », *in*: *Computer Graphics Forum*, vol. 20, 3, Wiley Online Library, 2001, pp. 174–183 (cit. on pp. 59, 83).

[Hu+19] Ping Hu et al., « Reducing simulator sickness with perceptual camera control », *in*: *ACM Transactions on Graphics (TOG)* 38.*6* (2019), pp. 1–12.

[Hua+16] Hui Huang et al., « Trip Synopsis: 60km in 60sec », *in*: *Computer Graphics Forum*, vol. 35, 7, Wiley Online Library, 2016, pp. 107–116 (cit. on pp. 37, 51).

[Hua+19a] Chong Huang et al., « Learning to capture a film-look video with a camera drone », *in*: *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 1871–1877 (cit. on p. 47).

[Hua+19b] Chong Huang et al., « Learning to film from professional human motion videos », *in*: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4244–4253 (cit. on p. 47).

[JH13] Jacek Jankowski and Martin Hachet, « A survey of interaction techniques for interactive 3D environments », *in*: *Eurographics 2013-STAR*, 2013 (cit. on p. 27).

[Jia+20] Hongda Jiang et al., « Example-driven virtual cinematography by learning camera behaviors », *in*: *ACM Transactions on Graphics (TOG)* 39.*4* (2020), pp. 45–1 (cit. on pp. 46, 50).

[Jia+21] Hongda Jiang et al., « Camera keyframing with style and control », *in*: *ACM Transactions on Graphics (TOG)* 40.*6* (2021), pp. 1–13 (cit. on p. 46).

[JLC20] Alberto Jovane, Amaury Louarn, and Marc Christie, « Topology-aware Camera Control for Real-time Applications », *in*: *Motion, Interaction and Games*, 2020, pp. 1–10 (cit. on pp. 40, 83).

[Jou+15] Niels Joubert et al., « An interactive tool for designing quadrotor camera shots », *in*: *ACM Transactions on Graphics (TOG)* 34.*6* (2015), pp. 1–11 (cit. on pp. 26, 40).

[Jou+16] Niels Joubert et al., « Towards a drone cinematographer: Guiding quadrotor cameras using visual composition principles », *in*: *arXiv preprint arXiv:1610.01691* (2016).

[KC08] Kaveh Kardan and Henri Casanova, « Virtual cinematography of group scenes using hierarchical lines of actions », *in*: *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, 2008, pp. 171–178.

[Kur+14] Christian Kurz et al., « Generating Realistic Camera Shake for Virtual Scenes », *in*: *JVRB-Journal of Virtual Reality and Broadcasting* 10.*7* (2014) (cit. on pp. 45, 50).

[LC08] Tsai-Yen Li and Chung-Chiang Cheng, « Real-time camera planning for navigation in virtual environments », *in*: *International Symposium on Smart Graphics*, Springer, 2008, pp. 118–129 (cit. on p. 42).

[LC12] Christophe Lino and Marc Christie, « Efficient composition for virtual camera control », *in*: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 2012, pp. 65–70 (cit. on pp. 26, 33, 34, 35, 55, 57, 61).

[LC15] Christophe Lino and Marc Christie, « Intuitive and efficient camera control with the toric space », *in*: *ACM Transactions on Graphics (TOG)* 34.*4* (2015), pp. 1–12 (cit. on pp. 26, 34, 35, 44, 46, 50, 53, 55, 57, 60).

[LCL18] Amaury Louarn, Marc Christie, and Fabrice Lamarche, « Automated staging for virtual cinematography », *in*: *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, 2018, pp. 1–10.

[Lin+10] Christophe Lino et al., « A real-time cinematography system for interactive 3D environments », *in*: *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 2010, pp. 139–148 (cit. on p. 39).

[Lin+11a] Christophe Lino et al., « Computational model of film editing for interactive storytelling », *in*: *International Conference on Interactive Digital Storytelling*, Springer, 2011, pp. 305–308.

[Lin+11b] Christophe Lino et al., « The director's lens: an intelligent assistant for virtual cinematography », *in*: *Proceedings of the 19th ACM international conference on Multimedia*, 2011, pp. 323–332 (cit. on p. 35).

[Lin+14] Christophe Lino et al., « How Do We Evaluate the Quality of Computational Editing Systems? », *in*: *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014 (cit. on p. 118).

[Lin13] Christophe Lino, « Virtual camera control using dynamic spatial partitions », PhD thesis, Université Rennes 1, 2013 (cit. on p. 39).

[Liu+18] Sikang Liu et al., « Search-based motion planning for aggressive flight in se (3) », *in*: *IEEE Robotics and Automation Letters* 3.*3* (2018), pp. 2439–2446 (cit. on p. 48).

[LMG10] Denise Lam, Chris Manzie, and Malcolm Good, « Model predictive contouring control », *in*: *49th IEEE Conference on Decision and Control (CDC)*, IEEE, 2010, pp. 6137–6142.

[Loq+21] Antonio Loquercio et al., « Learning high-speed flight in the wild », *in*: *Science Robotics* 6.*59* (2021), eabg5810 (cit. on pp. 48, 50).

[Lou+20] Amaury Louarn et al., « An interactive staging-and-shooting solver for virtual cinematography », *in*: *Motion, Interaction and Games*, 2020, pp. 1–6 (cit. on p. 25).

[LT17] Alan Litteneker and Demetri Terzopoulos, « Virtual cinematography using optimization and temporal smoothing », *in*: *Proceedings of the Tenth International Conference on Motion in Games*, 2017, pp. 1–6 (cit. on pp. 43, 50).

[Mar+11] Daniel Markowitz et al., « Intelligent camera control using behavior trees », *in*: *International Conference on Motion in Games*, Springer, 2011, pp. 156–167.

[Mar+14] Fabio Marton et al., « IsoCam: Interactive visual exploration of massive cultural heritage models on large projection setups », *in*: *Journal on Computing and Cultural Heritage (JOCCH)* 7.*2* (2014), pp. 1–24 (cit. on p. 30).

[Mar+97] Joe Marks et al., « Design galleries: A general approach to setting parameters for computer graphics and animation », *in*: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 389–400 (cit. on p. 30).

[Mas98] J.V. Mascelli, *The Five C's of Cinematography: Motion Picture Filming Techniques*, Silman-James Press, 1998, ISBN: 9781879505414, URL: https://books.google.fr/books?id=0gBMAQAAIAAJ (cit. on p. 31).

[MCB16] Billal Merabti, Marc Christie, and Kadi Bouatouch, « A virtual director using hidden markov models », *in*: *Computer Graphics Forum*, vol. 35, 8, Wiley Online Library, 2016, pp. 51–67.

[McC+09] James McCrae et al., « Multiscale 3D navigation », *in*: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 2009, pp. 7–14 (cit. on p. 29).

[MCR90] Jock D Mackinlay, Stuart K Card, and George G Robertson, « Rapid controlled movement through a virtual 3D workspace », *in*: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990, pp. 171–176 (cit. on p. 28).

[Mer+15] Billal Merabti et al., « Insight: An annotation tool and format for film analysis », *in*: *Eurographics Workshop on Intelligent Cinematography and Editing*, The Eurographics Association, 2015, p. 57.

[MK11] Daniel Mellinger and Vijay Kumar, « Minimum snap trajectory generation and control for quadrotors », *in*: *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 2520–2525.

[MMG12] Clément Moerman, Damien Marchal, and Laurent Grisoni, « Drag'n Go: Simple and fast navigation in virtual environment », *in*: *2012 IEEE Symposium on 3D user interfaces (3DUI)*, IEEE, 2012, pp. 15–18 (cit. on p. 29).

[Näg+17a] Tobias Nägeli et al., « Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization », *in*: *IEEE Robotics and Automation Letters* 2.*3* (2017), pp. 1696–1703 (cit. on pp. 43, 50).

[Näg+17b] Tobias Nägeli et al., « Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization », *in*: *IEEE Robotics and Automation Letters* 2.*3* (2017), pp. 1696–1703 (cit. on p. 52).

[Näg+17c] Tobias Nägeli et al., « Real-time planning for automated multi-view drone cinematography », *in*: *ACM Transactions on Graphics (TOG)* 36.*4* (2017), pp. 1–10 (cit. on pp. 43, 51).

[NO04] Dennis Nieuwenhuisen and Mark H Overmars, « Motion planning for camera movements », *in*: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 4, IEEE, 2004, pp. 3870–3876 (cit. on p. 83).

[Osk+09] Thomas Oskam et al., « Visibility transition planning for dynamic camera control », *in*: *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2009, pp. 55–65 (cit. on pp. 39, 40, 45, 49, 50, 83).

[Özy+17] Onur Özyeşil et al., « A survey of structure from motion*. », *in*: *Acta Numerica* 26 (2017), pp. 305–364 (cit. on p. 46).

[Pas+10] Erick B Passos et al., « Neuronal editor agent for scene cutting in game cinematography », *in*: *Computers in Entertainment (CIE)* 7.*4* (2010), pp. 1–17.

[Rat+09] Nathan Ratliff et al., « CHOMP: Gradient optimization techniques for efficient motion planning », *in*: *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009, pp. 489–494 (cit. on p. 48).

[RCU10] Roberto Ranon, Marc Christie, and Tommaso Urli, « Accurately measuring the satisfaction of visual properties in virtual camera control », *in*: *International Symposium on Smart Graphics*, Springer, 2010, pp. 91–102 (cit. on p. 33).

[Rey99] Craig W Reynolds, « Steering behaviors for autonomous characters », *in*: *Game developers conference*, vol. 1999, Citeseer, 1999, pp. 763–782 (cit. on pp. 37, 41, 67).

[RH16] Mike Roberts and Pat Hanrahan, « Generating dynamically feasible trajectories for quadrotor cameras », *in*: *ACM Transactions on Graphics (TOG)* 35.*4* (2016), pp. 1–11 (cit. on p. 40).

[Rit+08] Tobias Ritschel et al., « Imperfect shadow maps for efficient computation of indirect illumination », *in*: *ACM Transactions on Graphics* 27.*5* (2008) (cit. on p. 58).

[RK21] James Rucks and Nikolaos Katzakis, « CamerAI: Chase Camera in a Dense Environment using a Proximal Policy Optimization-trained Neural Network », *in*: *2021 IEEE Conference on Games (CoG)*, IEEE, 2021, pp. 1–8 (cit. on p. 47).

[Ron+15] Rémi Ronfard et al., « The prose storyboard language: A tool for annotating and directing movies », *in*: *arXiv preprint arXiv:1508.07593* (2015).

[Ron20] Rémi Ronfard, « Film Directing Techniques for Computer Games and Animation », *in*: *Computer Graphics Forum*, vol. xx, x, Wiley Online Library, 2020, pp. xxxx–xxxx (cit. on p. 23).

[Rot82] Scott D Roth, « Ray casting for modeling solids », *in*: *Computer graphics and image processing* 18.*2* (1982) (cit. on p. 58).

[RU14] Roberto Ranon and Tommaso Urli, « Improving the efficiency of viewpoint composition », *in*: *IEEE Transactions on Visualization and Computer Graphics* 20.*5* (2014), pp. 795–807 (cit. on pp. 26, 33, 49, 83, 90).

[RWS17] Gregory Rogez, Philippe Weinzaepfel, and Cordelia Schmid, « Lcr-net: Localization-classification-regression for human pose », *in*: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3433–3441 (cit. on p. 46).

[SAB11] Ekrem Serin, Serdar Adali, and Selim Balcisoy, « Entropy assisted automated terrain navigation using traveling salesman problem », *in*: *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, 2011, pp. 41–48.

[San+14] Cunka Bassirou Sanokho et al., « Camera Motion Graphs. », *in*: *Symposium on Computer Animation*, Citeseer, 2014, pp. 177–188 (cit. on p. 45).

[SC14] Cunka Bassirou Sanokho and Marc Christie, « On-Screen Visual Balance Inspired by Real Movies », *in*: *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014 (cit. on p. 33).

[SGS04] Karan Singh, Cindy Grimm, and Nisha Sudarsanam, « The ibar: A perspective-based camera widget », *in*: *Proceedings of the 17th annual ACM symposium on User interface software and technology*, 2004, pp. 95–98 (cit. on p. 28).

[SGS09] Nisha Sudarsanam, Cindy Grimm, and Karan Singh, « CubeCam: a screen-space camera manipulation tool. », *in*: *Computational Aesthetics*, 2009, pp. 65–71 (cit. on p. 28).

[SM98] Pierre OM Scokaert and David Q Mayne, « Min-max feedback model predictive control for constrained linear systems », *in*: *IEEE Transactions on Automatic control* 43.*8* (1998), pp. 1136–1142 (cit. on p. 84).

[Ste05] Josef Steiff, *The Complete Idiot's Guide to Independent Filmmaking*, 2005 (cit. on pp. 5, 15).

[TB09a] Roy Thompson and Christopher J Bowen, *Grammar of the Edit*, vol. 13, Taylor & Francis, 2009 (cit. on pp. 68, 72).

[TB09b] Roy Thompson and Christopher J Bowen, *Grammar of the Shot*, vol. 13, Taylor & Francis, 2009 (cit. on pp. 6, 16, 18, 31, 56, 61).

[TS91] Seth J Teller and Carlo H Séquin, « Visibility preprocessing for interactive walkthroughs », *in*: *ACM SIGGRAPH Computer Graphics* 25.*4* (1991).

[TUI17] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda, « Visual SLAM algorithms: A survey from 2010 to 2016 », *in*: *IPSJ Transactions on Computer Vision and Applications* 9.*1* (2017), pp. 1–11 (cit. on p. 46).

[Váz+01] Pere-Pau Vázquez et al., « Viewpoint selection using viewpoint entropy. », *in*: *VMV*, vol. 1, Citeseer, 2001, pp. 273–280 (cit. on pp. 49, 56).

[Vie+09] Thales Vieira et al., « Learning good views through intelligent galleries », *in*: *Computer Graphics Forum*, vol. 28, 2, Wiley Online Library, 2009, pp. 717–726 (cit. on p. 35).

[Wei+18] Zijun Wei et al., « Good view hunting: Learning photo composition from dense view pairs », *in*: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5437–5446 (cit. on p. 118).

[Wil78] Lance Williams, « Casting curved shadows on curved surfaces », *in*: *ACM Siggraph Computer Graphics* 12.*3* (1978) (cit. on p. 58).

[WP12] Andrew Woo and Pierre Poulin, *Shadow algorithms data miner*, CRC Press, 2012 (cit. on p. 58).

[Wu+18] Hui-Yin Wu et al., « Thinking like a director: Film editing patterns for virtual cinematographic storytelling », *in*: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14.*4* (2018), pp. 1–22.

[Xie+18] Ke Xie et al., « Creating and chaining camera moves for qadrotor videography », *in*: *ACM Transactions on Graphics (TOG)* 37.*4* (2018), pp. 1–13 (cit. on p. 37).

[Yan+17] Yiming Yang et al., « HDRM: A resolution complete dynamic roadmap for real-time motion planning in complex scenes », *in*: *IEEE Robotics and Automation Letters* 3.*1* (2017), pp. 551–558 (cit. on p. 38).

[Yeh+11a] I-Cheng Yeh et al., « Efficient camera path planning algorithm for human motion overview », *in*: *Computer Animation and Virtual Worlds* 22.*2-3* (2011), pp. 239–250 (cit. on pp. 36, 50, 51).

[Yeh+11b] I-Cheng Yeh et al., « Social-event-driven camera control for multicharacter animations », *in*: *IEEE Transactions on Visualization and Computer Graphics* 18.*9* (2011), pp. 1496–1510 (cit. on pp. 36, 37, 41, 51).

DOCTORAT BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Cinématographie virtuelle en temps réel pour le suivi des cibles

Mot clés : Cinématographie virtuelle, Infographie, Contrôle de caméra

Résumé : Le suivi cinématographique d'une cible mobile dans un environnement 3D dynamique reste un problème difficile. Le défi est d'assurer simultanément un faible coût de calcul, un bon degré de réactivité et une haute qualité cinématographique malgré les changements soudains de l'environnement. Le manuscrit est divisé en quatre chapitres. Dans le premier chapitre, nous avons présenté le contexte et les domaines des applications sous-jacentes considérées dans cette thèse, qui se concentre davantage sur le suivi en temps réel pour des applications telles que les applications 3D interactives. Ensuite, nous présentons l'état de l'art dans lequel nous introduisons les connaissances de base des techniques de planification et de contrôle des caméras. Ensuite, nous présentons deux approches pour résoudre le problème du suivi de cible en temps réel. Tout d'abord, au chapitre 3, nous avons proposé de nous appuyer sur un espace de caméra orienté vers le cadrage, l'espace torique, pour suivre deux cibles. Nous avons proposé un modèle d'anticipation et un modèle de mouvement de la caméra correspondant à l'espace toric et avons mené des expériences pour montrer l'efficacité de la méthode proposée avec anticipation. Dans la deuxième approche, au chapitre 4, nous avons abordé les limites de la première approche. Nous avons conçu un nouvel espace de caméra, l'espace d'animation de caméra. Nous avons proposé une métrique de qualité de trajectoire que nous évaluons en utilisant des techniques de traçage de rayons. Là encore, nous avons mené des expériences pour montrer l'efficacité de la méthode proposée. Dans le chapitre 5 de la thèse, nous concluons en donnant des directions pour les travaux futurs.

Title: Real-time Virtual Cinematography for Target Tracking

Keywords: Virtual cinematography, Computer graphics, Camera control

Abstract: Cinematic tracking of a moving target in a dynamic 3D environment remains a challenging problem. The challenge is to simultaneously ensure low computational cost, a good degree of responsiveness and high cinematic quality despite sudden changes in the environment and this thesis addresses these challenges. The manuscript is divided into four chapters. In the first chapter, we presented the background and domains of the underlying applications considered in this thesis, which focuses more on real-time tracking for applications such as interactive 3D applications. Next, we present the state of the art in which we introduce the basic knowledge of camera planning and control techniques. Next, we present two approaches to solving the real-time target tracking problem. First, in Chapter 3, we proposed to rely on a framing-oriented camera space, the torus space, to track two targets. We proposed an anticipation model and a camera motion model corresponding to the torus space and conducted experiments to show the effectiveness of the

proposed method with anticipation. In the second approach, in Chapter 4, we addressed the limitations of the first approach. We designed a new camera space, the camera animation space. We proposed a trajectory quality metric that we evaluate using ray tracing techniques. Again, we conducted experiments to show the effectiveness of the proposed method. In Chapter 5 of the thesis, we conclude by giving directions for future work.