Camera-on-rails: Automated Computation of Constrained Camera Paths

Quentin Galvane *
INRIA / LJK, Grenoble

Marc Christie
IRISA / University of Rennes I

Chrsitophe Lino
INRIA

Rémi Ronfard
INRIA / LJK , Grenoble

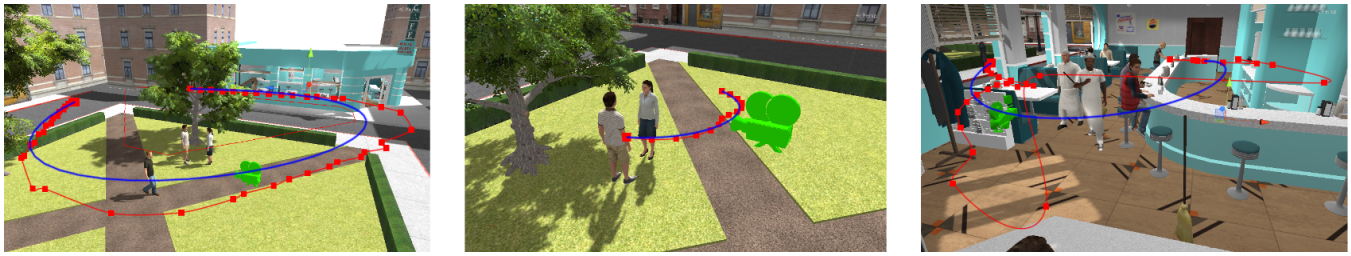


Figure 1: *Camera rails (in blue) are computed from a raw camera trajectory (in red) composed of optimal camera positions. The motion of the camera along this rail is then constrained and optimized for speed and orientation. The resulting camera motions provide natural tracking of characters as well as transitions between different camera viewpoints.*

Abstract

When creating real or computer graphics movies, the questions of how to layout elements on the screen, together with how to move the cameras in the scene are crucial to properly conveying the events composing a narrative. Though there is a range of techniques to automatically compute camera paths in virtual environments, none have seriously considered the problem of generating realistic camera motions even for simple scenes. Among possible cinematographic devices, real cinematographers often rely on camera rails to create smooth camera motions which viewers are familiar with. Following this practice, in this paper we propose a method for generating virtual camera rails and computing smooth camera motions on these rails. Our technique analyzes characters motion and user-defined framing properties to compute rough camera motions which are further refined using constrained-optimization techniques. Comparisons with recent techniques demonstrate the benefits of our approach and opens interesting perspectives in terms of creative support tools for animators and cinematographers.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.3.6 [Computer Graphics]: Methodology and Techniques—;

Keywords: virtual camera control, cinematography, virtual camera planning

1 Introduction

Recent improvement in the realism of computer games raises a requirement to provide users with a more cinematic experience. Particularly, with the possibility to share replays of a player's experience on the web, or to use game engines for creating movies (also known as *machinima*), there is a pressing demand for techniques to automatically generate virtual camera paths that capture what has occurred in the 3D scene. Furthermore, while to date most techniques have focused on filming dialogue-based scenes by relying on idioms, the problem of properly tracking character motions has been relatively under-addressed.

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

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

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

Ranging from the tracking of simple motions of characters to the creation of camera movements portraying complex motions of characters, our system provides means to efficiently compute stereotypical and natural camera shots through the simple specification of initial and final framings. As a result, our method allows to automatically create rushes that one can then use either as input to an editing or previsualisation system (*e.g.* similar to Galvane *et al.* [2015] or Lino *et al.* [2011]), or to create a single extended shot (long take) covering an entire movie scene.

The contributions of this paper are:

*e-mail: quentin.galvane@inria.fr

- a method to compute virtual camera rails that enable creating realistic camera transitions between an initial and a final framing, while accounting for the overall motion of target characters;
- a method to compute smooth camera paths constrained on a given rail. Our method nicely combines constraint solving and optimization to account for both aesthetic constraints (*e.g.* frame characters all along the camera path) and constraints on the camera (*e.g.* smoothly adapt the camera velocity and acceleration, and prevent the camera to get too close or too far from characters).

The paper is organized as follows. We first review previous research on virtual camera control (Section 2). We then provide an overview of our method (Section 3), and proceed with a detailed description of our two contributions (Sections 4 and 5). We further compare and discuss our results with several state of the art methods (Section 6) before concluding and offering perspectives for future work (Section 7).

2 Related Work

Optimization-based camera motions. Starting with optimization approaches [Drucker and Zeltzer 1994; Bares et al. 1998; Olivier et al. 1999], there has been a wealth of techniques proposed to automatically compute camera configurations (ie camera position, orientation and field of view) from the specification of user-defined visual properties such as visibility of targets, view angles, and relative or absolute on-screen position of targets. The camera parameters represent the search space, and techniques explore this space by minimizing cost functions expressed from the visual properties over the camera parameters. Unfortunately such solutions, which remain computationally expensive, are not easily extensible to compute camera paths for two reasons.

First, the computation of one optimal camera position per frame leads to the creation of jerky and overly reactive camera movements for which further smoothing or dampening models are required. A solution proposed by Assa *et al.* [Assa et al. 2008] consists in computing a global optimization of camera parameters over time for the whole animation. By creating a viewpoint entropy measure defined as the amount of motion of the 3D targets (characters) that projects onto the screen at each frame, the authors propose to maximize entropy over time and minimize the camera's internal energy. The problem of jerky camera motions is intrinsically addressed by encompassing speed and acceleration in this camera's internal energy function.

Second, if the optimal camera positions are computed for larger time steps and then further interpolated, there is the risk of not satisfying the visual properties in-between time steps (eg not maintaining the visibility of a target). Halper *et al.* [2000], for example, propose to extend their Camplan viewpoint computation technique by expressing properties over a camera path represented as a Bezier curve, and solving these properties at starting and ending points of the trajectory. Later, Christie *et al.* [2002] proposed the notion of hypertube – a parameterized camera motion – to express a sequence of traditional camera motions on which framing and view angle constraints are expressed. The overall trajectory is computed incrementally, solving a piece of trajectory and propagating continuity to other pieces. The sequence of camera motions is however selected manually and the computational cost remains important.

To address this cost issue, some novel camera representations have been introduced [Lino and Christie 2012; Lino and Christie 2015] that ensure the satisfaction of visual properties along time, by reducing the dimensionality of the search process compared to using classical camera models, and simplifying the computation of continuous camera motions. However, the technique uses the exact on-screen locations of targets to compute the viewpoint at every frame, and the tracking of targets that have jerky motions or sudden changes in positions will necessarily impact the camera motion accordingly, hence creating non-smooth trajectories.

Physically-based camera motions. A well-known technique to address jerky motions in computer graphics is to rely on physical models. Such models present the benefit of naturally smoothing trajectories and have been used in multiple camera navigation techniques [Hanson and Wernert 1997; Beckhaus 2002; Halper et al. 2001; Burelli and Jhala 2009]. More recently, Galvane *et al.* [2014] proposed to animate a camera by relying on physically-based steering behaviors to track characters. Lixandru *et al.* [2014] also proposed a physically realistic camera model (based on a real camera rig) to simulate a reactive camera capable of both tracking a moving target and producing plausible response to a variety of game scenarios. While benefits of such smoothing techniques are clear, the satisfaction of visual features in the simulation makes the solving process challenging, and the technique is mostly reserved to reactive contexts.

Path-planned camera motions. Path-planning techniques have been extensively exploited to create camera motions. Starting from Nieuwenhuisen and Overmars[2003] with probabilistic roadmap techniques to Oskam *et al.* [2009] using regular decompositions or Lino *et al.* [2010] proposing planning in dynamic viewpoint partitions, path-planning techniques display many interesting properties such as robustness, computational efficiency (in such low dimensions) or capacity to express and solve constraints along the camera path. Most solutions however require extra smoothing stages to deal with the regularity or irregularity of spatial decompositions. Furthermore, computed trajectories are not constrained to primitive camera motions, hence creating non natural camera motions in the context of cinematographic applications.

3 Overview

In this paper, we focus on the automated computation of natural camera motions between an initial and a final framing specified on one or two characters. A framing in this context represents the specification of the visual layout of characters on the screen in terms of exact on-screen positions, relative camera angle (high to low, front to back [Arijon 1976]) or on-screen sizes.

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

4 Building camera rails

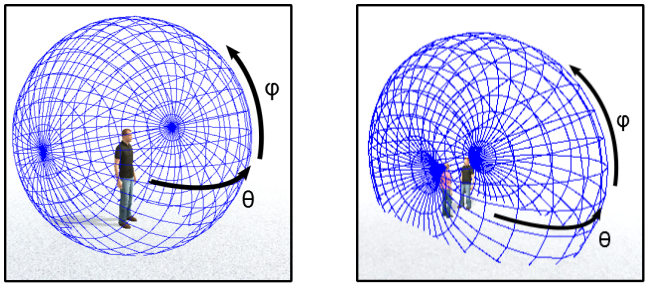
In order to build a camera rail, we first rely on a parametric representation of viewpoints similar to the one proposed in [Lino and Christie 2012]. This allows us to create a *raw trajectory* which satisfies, at each frame, the exact linearly interpolated framing between the initial framing and the final framing. This raw trajectory generally results in jerky camera motions. We then approximate

the *raw trajectory* with a Bezier curve so as to smooth it out. We restrict the complexity of the camera rail to a cubical curve, so as to limit rails to those that are commonly used in real movies, hence enhancing the naturalness of created shots.

4.1 Computing a raw trajectory

The input of this process is the motion of the characters as well as the user-specified initial and final framings –at times t_0 and t_1 respectively– of these characters. These framings comprise on-screen desired positions, on-screen sizes and vantage angles. Different optimization techniques can be used to compute actual camera configurations from these framing specifications; we here rely on [Lino and Christie 2015] that provides an efficient and algebraic implementation.

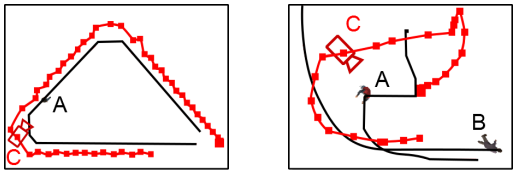
Initial and final camera configurations can therefore be expressed into a 2D-parametric representation, using one out of two types of manifold surfaces: a spherical surface (that can handle single-character configurations) or a toric-shaped surface (that can handle two-character configurations). In the case of a single character, the sphere is defined by using the shot size of the character. The horizontal and vertical vantage angles are defined in spherical coordinates, as shown in Figure 2a. The on-screen position of the character is finally determined through its projection onto the image plane. In the case of a pair of characters, we rely on the toric-shaped manifold surface proposed by Lino and Christie [2012] to compute the camera settings from the viewpoint. As explained by Lino and Christie, the toric surface is fully determined by the on-screen positions of the pair of characters. The vantage angles are then computed in a way similar to the single-character case, as shown in Figure 2b.



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

Figure 2: *Manifold surfaces used to defined visual properties in the case of (a) one target and (b) two targets. In both cases, profile and vantage angles are given respectively by θ and φ.*

To compute the *raw trajectory* linking the initial and final camera configurations, we propose to interpolate their framing properties (on-screen position, view angle and size) along time, and compute for each time step, the camera configuration satisfying the interpolated framing. Framing properties such as characters' on-screen positions, vantage angles or sizes are computed through a straightforward linear interpolation. In the case of a pair of characters, it is however required to distinguish two types of viewpoint interpolations. Indeed, since the vantage angle and the size of the target characters are correlated (getting closer to one character changes the vantage angle on the other), it is only possible to constrain (and thus interpolate) one of them. Figure 3 illustrates the computation of the raw path which interpolates between both viewpoints.



(a) **(b)**

Figure 3: *Example of raw trajectories computed in the cases of (a) a single moving target A and (b) two moving targets A and B. The camera raw trajectory is displayed in red while the targets' trajectories are displayed in black.*

4.2 From raw trajectories to camera rails

The computation of a camera rail then requires a curve fitting process to create a rail as close as possible to the raw trajectory (hence satisfying the framing properties along the path), while smoothing it out. While a classical B-spline model provides a good fitting, it may result in the creation of complex, hence unnatural, camera paths. Simpler models such as quadratic Bezier curves (as suggested by [Olivier et al. 1999] in their CamPlan system) remain limited (*e.g.* cannot handle loops). We here propose a cubic Bezier curve, which offers a simple representation of a camera rail with limited curvature changes and yet provides sufficient flexibility to handle most common camera motions from the literature [Arijon 1976].

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

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

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

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

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

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

Finally, as the maximum error is infinite, we know that the solution of the system corresponds to the minimum error. To compute control points P_1 and P_2, we then solve this system of linear equations. Figure 4 shows how our method approximates complex raw paths with simple camera rails.

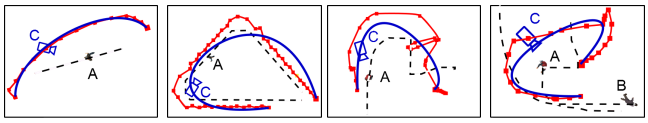


Figure 4: *Examples of camera rails (in blue) computed to approximate raw trajectories (in red) when tracking one or several characters with increasingly complex motions (from (a) to (d)). As the motion of the character becomes more complex, our method still provides a rail that well approximates the raw path.*

5 Moving the camera on the rail

We now focus on generating a smooth camera motion (in position and orientation) along this rail while maintaining an optimal framing over the target characters. To address this problem, we first compute at each time step t_i, the optimal camera position on the rail that satisfies the interpolated framing at t_i. We then improve this camera motion through an iterative optimization process which accounts for constraints on the camera (in terms of position, velocity and acceleration). Hence, we compute a camera motion closest to the optimal framing while enforcing smoothing constraints. We perform a similar process to compute the orientation of the camera along the path. These stages are described in the following sections.

5.1 Raw camera motion

To initiate a raw camera motion along the rail, we try at each time step to position the camera at an optimal position *on the rail*, knowing its previously defined raw position, *i.e. on the raw trajectory*. Though a straightforward solution could consist in projecting this raw camera position onto the rail, it may not ensure the satisfaction of visual properties (see Figure 5 for an example). Our solution consists in finding the position on the rail that is closest to the intersection between the rail and the manifold surface. Since every position on this manifold surface satisfies part of the desired visual composition (at least the on-screen position of target characters, as shown in [Lino and Christie 2012]), this methods tends to provide better camera positions to frame the target characters (and limits disturbing changes in on-screen sizes of characters).

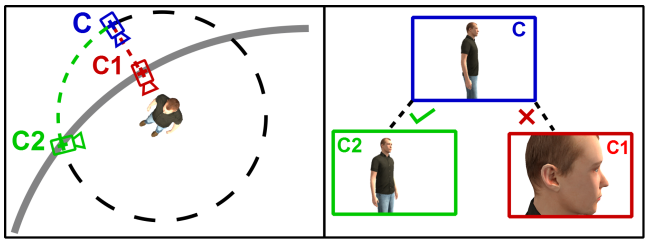


Figure 5: *The point on the rail (in grey) that provides the best approximation of the desired camera viewpoint C is not its projection $C1$ on the rail. It is given by the closest intersection $C2$ of the rail with the manifold surface (in black).*

As the intersection of a Bezier curve with a manifold surface is not straightforward, we compute the optimal camera position at a given frame by performing a dichotomous search on the neighborhood (along the rail) of the position found at previous frame. This search algorithm is detailed in Algorithm 1.

Algorithm 1 Dichotomous search of the point on the rail that is the closest to the manifold surface. The search is performed in the neighborhood (defined by Δ) of the previous optimal camera position p. $Rail(t)$ returns the position of the point at time t on the rail.

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

5.2 Smooth camera motion

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

The problem is solved using an optimization process which takes as input a shot duration d (comprising N frames) and a rail of length L. This process then minimizes equation (1), where x_i and x'_i are the parameters respectively defining the optimal and optimized camera position on the rail. Our optimization process is then subject to a number of constraints, each addressing one aspect of the motion. Firstly, through constraint (2) we state that the camera position will always belong to the rail, in-between both extremities of the rail. Secondly, through constraints (3) and (4) we state that the camera initial and final positions will be the extremities of the rail (P_0 and P_3 respectively). Thirdly, through constraints (5) and (6) we define a maximum velocity v_{max} and a maximum acceleration a_{max} for the camera motion. Finally, through constraints (7) and (8) we state that the camera motion will start with a zero speed and end with a zero speed (*i.e.* a complete stop of the camera). In other words, our optimization process is formulated as

Minimize

$$\sum_{i=0}^{N} |x'_i - x_i| \quad (1)$$

Subject to

$$0 \leq x'_i \leq 1 \quad (2)$$
$$x'_0 = 0 \quad (3)$$
$$x'_N = 1 \quad (4)$$
$$|x'_i - x'_{i-1}| \leq v_{max} * dt/L \quad \forall i \geq 1 \quad (5)$$
$$|2x'_{i-1} - x'_i - x'_{i-2}| \leq a_{max} * dt^2/L \quad \forall i \geq 2 \quad (6)$$
$$x'_1 - x'_0 \leq a_{max} * dt^2/L \quad (7)$$
$$x'_N - x'_{N-1} \geq -a_{max} * dt^2/L \quad (8)$$

Within this formalization, the problem can now be solved using

any existing linear programming library [1]. However, to ensure that our problem has a solution, we further define an implicit constraint on the minimum input values of v_{max} and a_{max} in the following way. If we use the minimum acceleration satisfying all constraints, the camera will constantly accelerate until it reaches half of the rail length at precisely half of the rush duration, then constantly decelerate until it reaches the end of the rail at precisely the end of the rush (with zero speed). This can be formalized in a simple mathematical way, as

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

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

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

5.3 Camera orientation

The two previous steps have computed a smooth camera motion (in terms of position) along the rail. Now computing the optimal camera orientation at each frame, given its position on the rail and a framing to satisfy, is easily addressed in [Lino and Christie 2012] through an algebraic formulation of camera orientation. However, for the same reasons as before, we also want to limit the angular speed and acceleration of the camera while it is moving along the rail to avoid jerky camera rotations that may occur. In a way similar to camera position, we therefore perform an optimization process along each of the three axes of the camera (*i.e.* pan, tilt, and roll).

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

Minimize

$$\sum_{i=0}^{N} |\theta'_i - \theta_i| \tag{9}$$

Subject to

$$\theta'_0 = \theta_0 \tag{10}$$

$$\theta'_N = \theta_N \tag{11}$$

$$|\theta'_i - \theta'_{i-1}| \leq \dot{\theta}_{max} * dt \quad \forall i \geq 1 \tag{12}$$

$$|2\theta'_{i-1} - \theta'_i - \theta'_{i-2}| \leq \ddot{\theta}_{max} * dt^2 \quad \forall i \geq 2 \tag{13}$$

$$|\theta'_1 - \theta'_0| \leq \ddot{\theta}_{max} * dt^2 \tag{14}$$

$$|\theta'_N - \theta'_{N-1}| \leq \ddot{\theta}_{max} * dt^2 \tag{15}$$

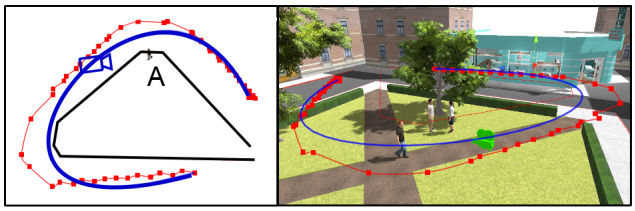
[1] we used GLPK (GNU Linear Programming Kit)

6 Results

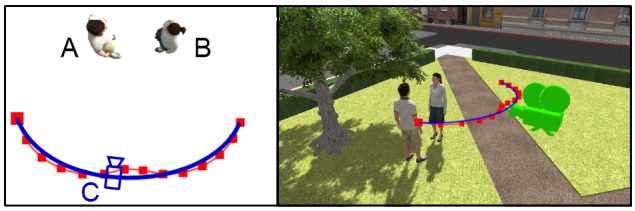
In this section we analyze the performance of our solution on a number of scenarios and compare our approach with existing camera control techniques on a publicly available dataset introduced by [Galvane et al. 2015]. All our results can be seen in the companion video (`https://cinematography.inria.fr/camera-on-rails`).

6.1 Performance

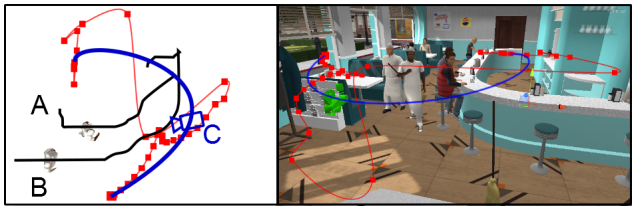
We here provide an overview of the performance of our method for different character placements and motions. The computational time required (on a Core i7@2.4GHz running Unity 5) to generate a camera rail is given in Table 1 for three different scenarios (S1 to S3). In each scenario, the camera makes a transition between two different user-defined framings. In the first scenario, the camera tracks a single character as he moves in the environment for 32 seconds. In the second scenario, the camera transitions between two different viewpoints specifications around a pair of static characters during 8 seconds. In the last scenario, the camera tracks during 18 seconds two moving characters walking at different speeds and with different walking directions. Figure 6 shows the characters' paths and the camera rails computed for each of these scenarios.



(a) *Scenario 1*



(b) *Scenario 2*



(c) *Scenario 3*

Figure 6: *Sample camera rails computed (a) for a single moving character, (b) for two static characters and (c) for two moving characters.*

Table 1 shows the computation time spent in each step of the process for each of the three scenarios. The overall computation time to create a smooth camera motion and orientation for the first scenario well illustrates the efficiency of our solution; less than 2 seconds are required to generate a 32-seconds camera motion. Moreover, most of computation time is spent in the optimization process (95% in average). Thus, there is not a significant impact of the number of targets or the type of transition on the overall computational time. We can however notice that the time spent in finding an optimal

position on the rail is greater when tracking a pair of targets than when tracking a single character, since working with toric surfaces is more expensive than working with simple spheres.

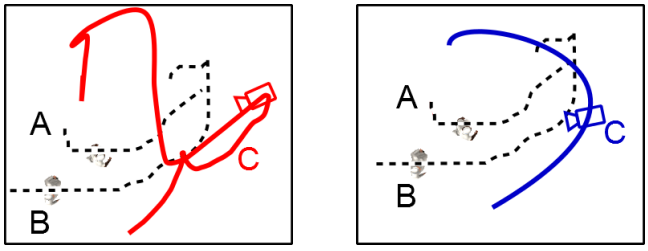
	S1	S2	S3
Raw path	5 ms	13 ms	10 ms
Bezier interpolation	13 ms	14 ms	15 ms
Desired positions on the rail	74 ms	103 ms	342 ms
Optimization of the position	920 ms	49 ms	231 ms
Desired orientation on the rail	7 ms	9 ms	26 ms
Optimization of the orientation	940 ms	117 ms	877 ms
Overall computation	1.95 s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

Table 1: *Computation times for the different steps of the planning process in three different scenarios (S1 to S3). S1: track a single moving character during 28s. S2: move around a pair of static characters for 8s. S3: track two moving characters for 20s.*

6.2 Comparison with other methods

We compared our method with three other camera planning techniques: (i) an approach relying on the model introduced by Lino and Christie [2012], (ii) a recent technique based on steering behaviors [Galvane et al. 2014], and (iii) a derived version of [Galvane et al. 2014] where the camera is steered along a virtual rail rather than optimized. All the comparisons are performed on the scenario S3.

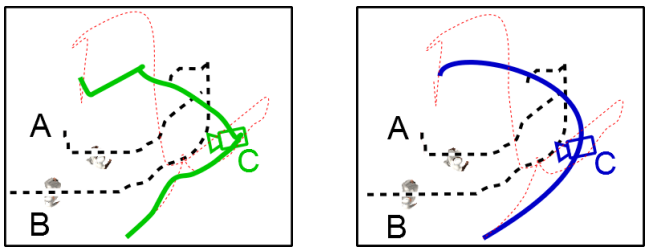
Lino and Christie [2012]. Their technique can be used to create camera motions that strictly enforce a simple framing on a pair of targets. The resulting camera path corresponds to our raw camera path. The main problem of this method is illustrated in Figure 7. When the targets' motions are too complex, the method will tend to create jerky and unnatural camera motions. Moreover, as the camera viewpoint is recomputed at each frame without considering either the previous camera position, nor the camera speed, the resulting camera motion is not guaranteed to be continuous nor smooth.



(a) (b)

Figure 7: *Comparison between the trajectory computed with (a) the Manifold surface introduced by Lino and Christie and (b) our camera rail.*

Galvane *et al.* [2013; 2014] Based on the model of Lino and Christie, Galvane *et al.* introduced a physically based model for camera control that limits camera motions to ensure physically plausible motions. In their approach, an optimal viewpoint is computed and the camera is then steered toward this viewpoint. The resulting camera motions are much smoother than the one obtained with [Lino and Christie 2012] and provide interesting results in simple cases. However, as shown in Figure 8, when confronted to complex situations, steering-based cameras fail in generating smooth trajectories. Furthermore, the computational time required to compute the camera path from the raw trajectory by using steering behaviors is greater than when using our method. It takes 3.33 seconds to compute both the position and orientation for the whole duration of the rush (given the raw optimal trajectory), whereas when using our optimization process it only takes 1.1 second. Even though our approach only offers an off-line solution – it requires an analysis of the target characters' motions –, it demonstrates better performances in average than when using Galvane *et al.* real-time method.



(a) (b)

Figure 8: *Comparison between the path computed (a) by steering a camera and (b) by using our rail positioning process.*

Extended version of Galvane *et al.* [2014] We also compared our solution to an extended version of the method of Galvane *et al.* . Instead of freely steering the camera around in the environment, we constrained the camera motion along the same rail as in our method. We then steered, at each frame, the camera towards the optimal viewpoint on the rail by using steering behaviours defined in Galvane *et al.* [2014]. This last comparison is important since it confirms the necessity of the optimization we propose. Indeed, Figure 9 shows the evolution of the position of the camera on the rail for both methods and displays the optimal camera position along time. Our method closely follows the optimal position whereas the steering camera is always behind on the rail.

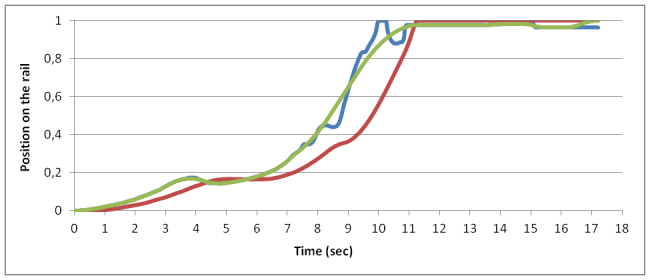


Figure 9: *Position of the cameras on the rail along time. Our optimized solution (green) smoothly approximates the optimal camera positions on the rail (blue) while the autonomous camera (red) hardly keeps-up and introduced a delay.*

This graph reveals the main drawback of using a reactive method like steering behaviors. The camera only moves or accelerates at the last possible moment for it does not anticipate the characters motion. Our solution on the other hand is able to anticipate the movements by globally optimizing the camera positions. Figure 10 illustrates the difference between the two approaches. When the characters suddenly move away from the camera, the reactive solution does not handle the abrupt movement and looses the visibility over the targets for a moment. Our camera is able to maintain the visibility over the characters by moving earlier along the rail.

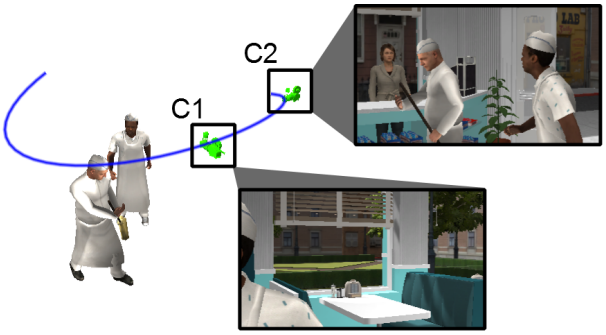


Figure 10: *Viewpoints of the two constrained cameras as the characters move. The autonomous camera $C1$ is not able to track them whereas our solution $C2$ is able to maintain the visibility over the two characters by anticipating their movements.*

7 Conclusion

In this paper, we have introduced a novel approach to create smooth and natural camera motions that relies on traditional cinematographic techniques. Our technique automatically computes a camera rail from the specification of initial and final framings, and by knowing the motions of the targets to track. We rely on two constrained-optimization processes that ensure both the proper framing of target characters, and smooth changes in the camera speed. Results demonstrate the benefits of our technique in comparison with recent approaches. Possible extensions of our method could be to constrain these camera motions to existing physical camera rigs to improve realism and to add extra constraints along the path at given key frames.

The solution we provide addresses a problem hardly explored before – the creation of cinematographic and smooth camera motions in virtual cinematography – with possible applications for the creation of tools to assist cinematographers and animators in rapidly drafting camera motions, and for automatic computation of cinematic replays in computer games.

References

ARIJON, D. 1976. *Grammar of the Film Language*. Hastings House Publishers.

ASSA, J., COHEN-OR, D., YEH, I.-C., AND LEE, T. 2008. Motion Overview of Human Actions. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2008 27*, 5 (December).

BARES, W. H., GREGOIRE, J. P., AND LESTER, J. C. 1998. Realtime Constraint-Based cinematography for complex interactive 3D worlds. In *Proceedings of AAAI-98/IAAI-98*, 1101–1106.

BECKHAUS, S. 2002. *Dynamic Potential Fields for Guided Exlporation in Virtual Environments*. PhD thesis, Fakultät für Informatik, University of Magdeburg.

BURELLI, P., AND JHALA, A. 2009. Dynamic artificial potential fields for autonomous camera control. In *Proceedings of the 5th AAAI Conference On Artificial Intelligence In Interactive Digitale Entertainment Conference*, AAAI Press.

CHRISTIE, M., LANGUÉNOU, E., GRANVILLIERS, L., AND LANGUÉNOU, E. 2002. Modeling Camera Control with Constrained Hypertubes. In *Principles and Practice of Constraint Programming*, vol. 2470 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 618–632.

DRUCKER, S. M., AND ZELTZER, D. 1994. Intelligent camera control in a virtual environment. In *Graphics Interface*, 190–199.

GALVANE, Q., CHRISTIE, M., RONFARD, R., LIM, C.-K., AND CANI, M.-P. 2013. Steering Behaviors for Autonomous Cameras. In *Motion in Games*, ACM, 93–102.

GALVANE, Q., RONFARD, R., CHRISTIE, M., AND SZILAS, N. 2014. Narrative-Driven Camera Control for Cinematic Replay of Computer Games. In *Motion In Games*, ACM, Los Angeles, United States.

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

HALPER, N., AND OLIVIER, P. 2000. CAMPLAN: A Camera Planning Agent. In *Smart Graphics 2000 AAAI Spring Symposium*, 92–100.

HALPER, N., HELBING, R., AND STROTHOTTE, T. 2001. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. In *Proceedings of the Eurographics Conference (EG 2001)*, Computer Graphics Forum, vol. 20, 174–183.

HANSON, A., AND WERNERT, E. 1997. Constrained 3d navigation with 2d controllers. In *Proceedings of the IEEE Visualization Conference (VIS 97)*, 175–182.

LINO, C., AND CHRISTIE, M. 2012. Efficient Composition for Virtual Camera Control. In *ACM Siggraph / Eurographics Symposium on Computer Animation*, P. Kry and J. Lee, Eds.

LINO, C., AND CHRISTIE, M. 2015. Intuitive and Efficient Camera Control with the Toric Space. *ACM Transactions on Graphics (TOG). Proceedings of ACM SIGGRAPH 2015*.

LINO, C., CHRISTIE, M., LAMARCHE, F., SCHOFIELD, G., AND OLIVIER, P. 2010. A Real-time Cinematography System for Interactive 3D Environments. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 139–148.

LINO, C., CHRISTIE, M., RANON, R., AND BARES, W. 2011. The Director's Lens: An Intelligent Assistant for Virtual Cinematography. In *ACM International Conference on Multimedia*.

LIXANDRU, E. T., AND ZORDAN, V. 2014. Physical Rig for First-person, Look-at Cameras in Video Games. In *Motion in Games*, ACM.

NIEUWENHUISEN, D., AND OVERMARS, M. H. 2003. Motion planning for camera movements in virtual environments. Tech. Rep. UU-CS-2003-004, Institute of Information and Computing Sciences, Utrecht University.

OLIVIER, P., HALPER, N., PICKERING, J. H., AND LUNA, P. 1999. Visual Composition as Optimisation. In *Artificial Intelligence and Simulation of Behaviour*, 22–30.

OSKAM, T., SUMNER, R. W., THUEREY, N., AND GROSS, M. 2009. Visibility transition planning for dynamic camera control. In *Proc. SIGGRAPH/Eurographics Symp. on Computer Animation*, 55–65.