Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2014)
Vladlen Koltun and Eftychios Sifakis (Editors)

# Camera Motion Graphs

C. Sanokho[1], C. Desoche[1], B. Merabti[1,2], T-Y Li [3] and M. Christie[1]

[1]IRISA, University of Rennes 1, France
[2]Algiers Polytechnic School, Algeria
[3]National Chengchi University, Taiwan

**Abstract**

*This paper presents Camera Motion Graphs, a technique to easily and efficiently generate cinematographic sequences in real-time dynamic 3D environments. A camera motion graph consists of (i) pieces of original camera trajectories attached to one or multiple targets, (ii) generated continuous transitions between camera trajectories and (iii) transitions representing cuts between camera trajectories. Pieces of original camera trajectories are built by extracting camera motions from real movies using vision-based techniques, or relying on motion capture techniques using a virtual camera system. A transformation is proposed to recompute all the camera trajectories in a normalized representation, making camera paths easily adaptable to new 3D environments through a specific retargeting technique. The camera motion graph is then constructed by sampling all pairs of camera trajectories and evaluating the possibility and quality of continuous or cut transitions. Results illustrate the simplicity of the technique, its adaptability to different 3D environments and its efficiency.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

## 1. Introduction

With significant advances in the quality of real-time rendering techniques, there is a pressing demand to properly convey complex 3D contents through appropriate cinematography (cinematography is here understood as positioning, moving and cutting between cameras). Typically, computer games increasingly rely on elements of style/genre drawn from real movies in terms of camera placements, trajectories and edits. And in the game industry, these issues are mostly addressed through collections of manually crafted camera animations and edits.

The path towards a fully automated computation of cinematography is complex. Current contributions address the problem by procedurally generating camera trajectories. Using optimization or path planning techniques, paths are synthesized accounting for velocity [NO03], visibility of targets [OSTG09, CNO12] or visual properties to be ensured along the trajectory [HHS01]. However, most trajectories still display a distinguishable synthetic aspect. Two reasons: through many years of training, audience is now educated to certain types of camera trajectories pertained to the devices and techniques used in the real movies. Yet most generative models of trajectories do not account for such techniques. And second, there are specificities and subtleties on camera motions which are difficult to reproduce with generative models: aesthetic variations in speed, or noise in motions due to real camera devices or hand-held cameras. These speed variations or noises are part of the quality of camera trajectories and participate in their realism.

A solution to reproduce such effects would consist in reusing real camera paths, which intrinsically contain these variations and noise. But how to adapt, or *retarget*, these existing paths to new 3D environments? One first needs to reproduce the proper framing of targets (*i.e.* on-screen locations of targets throughout the trajectory) with scenes and target motions different from the original one. Second the paths need to be adapted to the specific scales and target positions of the new 3D environment. And third, the visibility of targets must be evaluated in the new environment and strategies must be developed to avoid occlusions.

Our goal in this paper is to retain the realism of camera paths by extracting them from real data (existing movies or

DOI: 10.2312/sca.20141136

motion captured data), and to provide a means to retarget these paths to new 3D environments. Directly inspired by the way character animation techniques strongly rely on motion captured information to ensure realism [Zhe13], we propose *camera motion graphs*. A camera motion graph is a motion graph [KGP02] in which pieces of camera trajectories are connected through continuous transitions (denoted $T^C$) or non-continuous transitions (denoted $T^N$) from one trajectory to another. The construction of a camera motion graph consists in sampling all pairs of camera trajectories in order to evaluate the possibility and the cost of transitions. The cost of a transition is defined using cinematographic continuity rules (see [Tho09]).

This paper describes the three stages in camera motion graphs: (i) extracting real camera trajectories and expressing them in a normalized representation, (ii) constructing a camera motion graph by building possible transitions between trajectories, and (iii) applying the camera motion graph in a new 3D environment.

The benefits of our approach are: (i) the simplicity in re-using real content inside a camera motion graph structure, (ii) the retargeting of camera paths in new 3D environments and (iii) the computational efficiency. To demonstrate the benefits, we illustrate our approach on examples involving multiple characters in cluttered 3D environments. Practical applications of this work can be found in game industry, but also for animation studios wishing to re-use carefully crafted trajectories, and previzualisation industry to explore possible trajectories or edits of a cinematographic sequence within a few minutes.

## 2. Related Work

Controlling a virtual camera is a complex problem, not due to the dimensionality of the problem (a camera is generally modeled with 7 degrees of freedom), but to the properties it should ensure (speed, angle, visibility of targets, framing of targets and maintaining on-screen composition of targets). A survey on the different techniques has been proposed by Christie *et al.* [CON08]. We here restrict the study to issues addressed in this paper: planning camera paths and edits between paths.

Largely based on techniques from the field of robotics, Nieuwenhuisen and Overmars [NO03] propose to precompute a probabilistic roadmap (PRM) in the free space around a static 3D model. The roadmap is then used to generate a synthetic path from user-specified initial and final camera positions. The synthesized path is smooth and optimized by considering maximum angular rotations on the camera parameters, and enables the user to automatically navigate around the model.

Oskam *et al.* [OSTG09] later extend the use of roadmaps using a precomputed regular decomposition of an environment. The roadmap is computed on the entire free space of the scene and paths are automatically computed to transit from one specified viewpoint to another while maintaining visibility of static or dynamic targets. The authors propose to divide the scene in cells and fill the cells with spheres. The intersection of two spheres is a portal through which the camera can move. The final connection between the portals are set and refined by sampling the visibility between two portals using stochastic ray casting. The camera trajectory is then computed from a point A to a point B using the roadmap and smoothing the path through portals by using real-time rendering techniques for visibility.

Li *et al.* [LC08] also propose to use roadmaps to track dynamic targets. Interestingly, the roadmap is computed in the local reference frame of the dynamic target, rather than of the environment. This roadmap structure is updated as the scene evolves using a lazy evaluation scheme to check whether way-points in the roadmap still ensure the visibility of the target. Furthermore, the authors introduced virtual links in the roadmap which allows the camera to perform cuts (a non-continuous transition) between two positions. The approach however focuses on a single target.

Other well-known planning techniques such as corridor maps have been used for planning camera motions (see [VMGL12]) for the specific tasks of tracking a large number of targets. Using roadmaps for filming dynamic targets is also a path following by Lino *et al.* [LC12a]. The solution consists in first decomposing the 3D environment using spatial viewpoint partitions into a Binary Space Partition (BSP) representation. The viewpoint partitions are modeled after cinematographic properties. The technique then performs a BSP edge sampling to construct a camera navigation graph, and performs path planning in the graph by using the cinematographic properties. The spatial partitions are recomputed at a frequency lower than the frame rate to adapt to dynamic targets.

In order to avoid both the cost of planning camera motions and the low realism of synthetic trajectories, the solution we propose consists in re-using and adapting real camera paths. This option has not been explored by previous contributions, with two notable exceptions: [SB04] and [KRE*10].

In tackling the issue of re-using camera paths from previous navigations for improving the exploration tasks of a 3D scene, Singh and Balakrishnan [SB04] propose an approach to store and efficiently extract paths in space and in time using clustering techniques. The approach is however dedicated to this specific task and does not address editing issues nor realism in the paths.

More recently, Kurz *et al.* [KRE*10] propose to improve the realism of computer generated trajectory by adding noise extracted from real trajectories. This method is a three-step process. First, a database of noise is created. The real camera trajectories are filtered using a Taubin filter. The noise (also called details), is obtained by applying a Gabor transform to the difference between the base and the real camera trajectory. The noise is then re-applied to a synthetic trajectory. This method shows that real data can be used in a smart way to improve computer generated trajectories. However this approach only extracts the details of the trajectory and not the overall motion.

Our objective in this paper is to extract real camera motions from different sources and retarget them to new contents (*i.e.* new targets in new 3D environments). We furthermore propose to organize these trajectories in a motion graph structure to enable possible transitions or cuts between the adapted trajectories.

## 3. Camera Motion Graphs

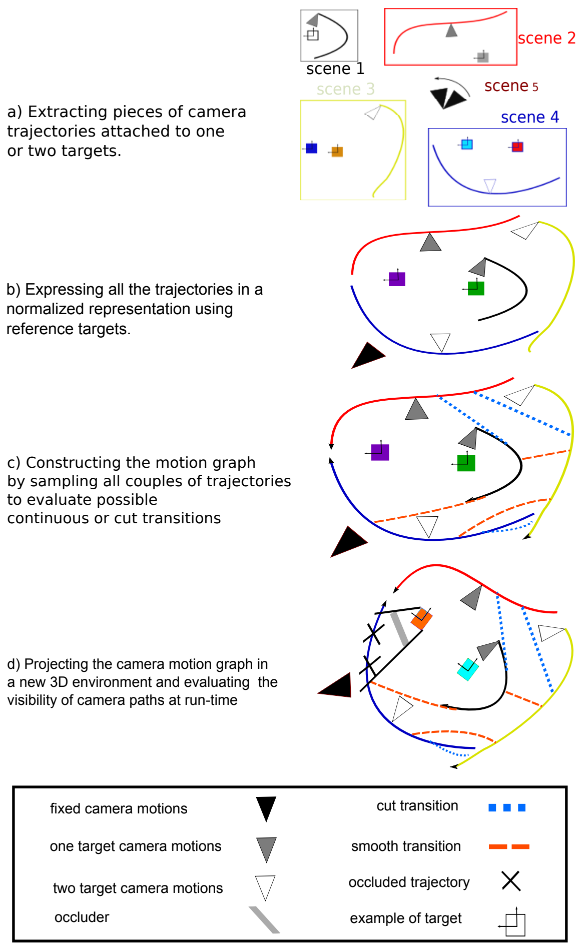

scene 1
scene 2
scene 3
scene 5
scene 4
a) Extracting pieces of camera trajectories attached to one or two targets.
b) Expressing all the trajectories in a normalized representation using reference targets.
c) Constructing the motion graph by sampling all couples of trajectories to evaluate possible continuous or cut transitions
d) Projecting the camera motion graph in a new 3D environment and evaluating the visibility of camera paths at run-time
fixed camera motions
one target camera motions
two target camera motions
occluder
cut transition
smooth transition
occluded trajectory
example of target

**Figure 1:** *Overview of the process consisting in constructing a camera motion graph from real trajectories, and exploiting the camera motion graph in a new 3D environment.*

Our camera motion graph is a directed graph [KGP02, LCR*02]. Each node corresponds to a motion clip (i.e. a piece of a camera trajectory) expressed in a normalized representation. Each edge in the motion graph corresponds either to a continuous transition or a cut, from one motion clip to another.

Creating and using camera motion graphs follows a 3-stage process. The first stage consists in extracting camera trajectories from real film footage or using motion capture techniques, and then expressing the trajectories in a normalized representation (see Section 4). The second stage consists in constructing the motion graph by computing the possible transitions between trajectories through the evaluation of two costs: a cost for continuous transitions and a cost for non-continuous transitions (see Section 5). And finally, the third stage consists in generating a camera path in real-time by using the camera motion graph structure as the scene dynamically evolves (see Section 6). This stage typically handles the issue of target occlusion by adapting the camera path to deal with small occlusions, or by selecting a transition between paths to deal with large occlusions.

The input of our system is a 3D scene encompassing one or multiple dynamic targets. The animations of the scene or the targets are not known beforehand. A simple semantic layer is required to contextually describe the actions performed by the targets. The real-time output of the system is a camera path adapted to both the context of the scene and its 3D representation while ensuring the visibility of targets.

## 4. Extracting Camera Trajectories

Camera trajectories extracted from tracking tools or motion capture software are usually expressed using Cartesian coordinates defined in the global basis of the 3D scene (for the position) and quaternions (for the orientation). In order to express these trajectories in a normalized representation, both suitable for retargeting and insertable in a camera motion graph, a better representation is required.

### 4.1. Representation of camera trajectories

Let us first define some notations. Let $\mathbf{q}$ be a camera configuration. A camera configuration is defined by a Cartesian position $\mathbf{x_q}$, a quaternion orientation $\mathbf{u_q}$, a field of view $\gamma_\mathbf{q}$, and a depth of field $\rho_\mathbf{q}$. A camera trajectory $\mathcal{T}$ is represented by a sequence of $n$ camera configurations, at the rate of one camera configuration per frame (at 30 fps). A configuration $\mathbf{q}(i)$ represents a camera configuration at frame $i$ in a trajectory. Let $\mathbf{o}$ be a target in the scene defined by a Cartesian position $\mathbf{x_o}$, a quaternion orientation $\mathbf{u_o}$ and a height $h_\mathbf{o}$ (the height of a target is the size of its bounding box along axis $z$, in a $z$-up reference frame). A framing configuration $\mathbf{f^o}$ is a 2D position representing the on-screen projection of a target object position $\mathbf{x_o}$ through a camera configuration $\mathbf{q}$.

We propose to identify three groups of camera trajectories. The first (denoted $G_f$) only encompasses pan and tilt motions for fixed cameras (*i.e.* cameras with fixed positions that only swivel around horizontal or vertical axes), focusing on a single target (or a group of targets abstracted as a single target). The second (denoted $G_1$) gathers camera motions that focus on a single target (or a group abstracted as single target). Example trajectories are turning around a character or performing a traveling while following a character. And the third group ($G_2$), gathers camera trajectories defined over two or more targets. Example trajectories are circular motions around the characters, traveling motions towards the characters or crane motions. All three groups are represented in Figure 2.

We propose to express the trajectories in the reference frame of the targets so as to create a normalized representation. We refer to these targets as *reference targets*. Expressing trajectories in the reference frame of a single target is straightforward since a spherical camera coordinate system can be used to encode position, and quaternions for orientation. The problem is however more complex with two reference targets (see Section 4.1.3), but can be addressed by exploiting the manifold representation proposed by [LC12b].

Furthermore, rather than storing a sequence of camera orientations $\mathbf{u_q}$ that frame the reference target(s), we propose to store a sequence of framing configurations $\mathbf{f^o}(i)$ for each target $o$ (i.e. where the reference target $o$ is projected on the screen at frame $i$). The benefit of this representation is to retarget the camera orientations by maintaining the framing positions of new targets on the screen as they were recorded in the initial trajectory. Indeed, a simple replay of the camera orientations $\mathbf{u_q}$ independently of the target's motions would lead to empty shots or awkward framings as soon as the motions of the new targets do not correspond to the motions of the reference targets (which will often be the case).

### 4.1.1. Fixed camera motions $G_f$

A panoramic motion of the camera (see Figure 2(a)) is a fixed camera motion generally used to follow target objects. In order to re-use such a motion in a different environment, we need to (i) ensure that the camera stays at a fixed position, and (ii) that the camera frames the new target in the same way it framed the reference target. Therefore, in the group of fixed camera motions $G_f$, we choose to express the position $\mathbf{x}_q$ of a camera configuration $\mathbf{q}$ in the local basis $\mathbf{o}_0$ of the target $\mathbf{o}$ at its first frame ($\mathbf{o}(0)$ representing the target configuration at frame 0). The camera's orientation is expressed using a framing configuration $\mathbf{f^o}$ at frame $i$ (projected position of target $\mathbf{o}_i$ on the screen). Field of view and depth of field at each frame are the same as in the extracted trajectory. In order to address the scaling issue (dealing with a new target which size is different from the reference target), we normalize the representation by considering the height $h_\mathbf{o}$ of the reference target to be 1, and applying a scaling factor $s$ on the local coordinates of the camera so that the projected height of the target is maintained.

When reusing this trajectory in a new 3D scene with a new target object $\mathbf{o}'$, the newly computed camera configuration $\mathbf{q}'$ will be expressed at frame $i$ as:

$$\begin{cases} \mathbf{x}_{\mathbf{q}'}(i) &=& \mathbf{x}_{\mathbf{o}'}(0).\mathbf{u}_{\mathbf{o}'}(0).\mathbf{u}_\mathbf{o}(0).(h_{\mathbf{o}'}/h_\mathbf{o}).\mathbf{x}_\mathbf{o}(0) \\ \mathbf{u}_{\mathbf{q}'}(i) &=& \mathbf{u}_\mathbf{f}(i) \\ \gamma_{\mathbf{q}'}(i) &=& \gamma_\mathbf{q}(i) \\ \rho_{\mathbf{q}'}(i) &=& \rho_\mathbf{q}(i) \end{cases}$$

where $\mathbf{u_f}(i)$ represents the camera orientation computed from the framing configuration $\mathbf{f^o}(i)$ now targeting object $\mathbf{o}'(i)$. The camera location therefore remains fixed, while the camera swivels to maintain the recorded composition with new target $\mathbf{o}'$.

### 4.1.2. Single target motions $G_1$

In the group of trajectories $G_1$ (see Figure 2(b)), we represent the position of a camera configuration $\mathbf{q}(i)$ in a trajectory $\mathcal{T}$ with a spherical coordinate system defined in the local basis of the target $\mathbf{o}$ at frame $i$. This means that the camera motion is locally defined in the basis of the reference object, and that as the new target moves, the camera will move accordingly (unlike motions $G_f$). The camera orientation is expressed with a framing configuration $\mathbf{f^o}$ defined at frame $i$ on target $\mathbf{o}$. Field of view and depth of field at each frame are the same as in the extracted trajectory. The newly expressed camera configuration is therefore defined as $[\phi, \theta, d, \mathbf{f^o}, \gamma, \rho]^T$ where $\phi, \theta, d$ represents the spherical coordinates of camera configuration $\mathbf{q}$ in the basis of reference target $\mathbf{o}$. A normalization is also computed to address the scaling issue.

When re-using this trajectory with a new target object $\mathbf{o}'$, the camera configuration $\mathbf{q}'$ will be computed at every frame $i$ as:

$$\begin{cases} \mathbf{x}_{\mathbf{q}'}(i) &=& \mathbf{x}_{\mathbf{o}'}(i).\mathbf{u}_{\mathbf{o}'}(i).h_{\mathbf{o}'}/h_\mathbf{o}.S(\phi(i),\theta(i),d(i)) \\ \mathbf{u}_{\mathbf{q}'}(i) &=& F(\mathbf{f^o}(i),\mathbf{o}'(i)) \\ \gamma_{\mathbf{q}'}(i) &=& \gamma_\mathbf{q}(i) \\ \rho_{\mathbf{q}'}(i) &=& \rho_\mathbf{q}(i) \end{cases}$$

where $S(\phi(i), \theta(i), d(i))$ computes at frame $i$ the transformation from spherical coordinates $(\phi, \theta, d)$ to Cartesian coordinates, and $F(\mathbf{f^o}(i), \mathbf{o}'(i))$ computes the camera orientation from the framing configuration $\mathbf{f^o}$ at frame $i$ now targeting object $\mathbf{o}'$ at frame $i$.

### 4.1.3. Multiple target motions $G_2$

Finding a normalized representation in which to express different trajectories that involve two reference targets is not straightforward, and possible solutions come with limitations. In this paper, we propose to rescale each trajectory. The scaling is applied so as to normalize the distance between the two targets. Rather than expressing the trajectories in the basis of one of the targets, we propose to use the manifold coordinate system proposed by Lino *et al.* [LC12b] in a way that enables easy retargeting. The manifold coordinate system was initially defined to maintain the on-screen locations of two or three tracked dynamic targets (see Figure 12). Indeed, given the desired on-screen location of two targets and the 3D positions of these targets in the scene, the authors proposed to compute a manifold surface on which each and every point ensures the desired on-screen locations. Our principle here is to derive this representation by expressing our camera trajectories for two targets using a sequence of manifold coordinates on different manifold surfaces (each specified composition with two targets represents a manifold surface, and a viewpoint is a point on this surface). See Appendix A for details.

We therefore express the camera positions of a trajectory $\mathcal{T}$ using a manifold coordinate representation defined over a couple of targets $o_1$ and $o_2$. The camera orientation is specified using two composition configurations $\mathbf{f}^{\mathbf{o}_1}$ and $\mathbf{f}^{\mathbf{o}_2}$ (see Figure 2(c)). The newly expressed configuration is defined by $[\varphi, \theta, \alpha, \mathbf{f}^{\mathbf{o}_1}, \mathbf{f}^{\mathbf{o}_2}, \gamma, \rho]^T$ where $\varphi, \theta, \alpha$ are the manifold coordinates of the camera configuration $\mathbf{q}$ and the targets positions $o_1$ and $o_2$ at a given frame $i$.

When re-using this trajectory with a new couple of target objects $\mathbf{o}'_1$ and $\mathbf{o}'_2$, the camera configuration $\mathbf{q}'$ will be computed at each frame $i$ as:

$$\begin{cases} \mathbf{x}_{\mathbf{q}'}(i) & = & M^{\alpha}_{x}(\varphi(i), \theta(i)) \\ \mathbf{u}_{\mathbf{q}'}(i) & = & M^{\alpha}_{q}(\varphi(i), \theta(i)) \\ \gamma_{\mathbf{q}'}(i) & = & \gamma_{\mathbf{q}} \\ \rho_{\mathbf{q}'}(i) & = & \rho_{\mathbf{q}} \end{cases}$$

where $M^{\alpha}_{x}(\varphi(i), \theta(i))$ computes the Cartesian coordinates of the manifold coordinates $(\varphi(i), \theta(i))$ associated with desired composition $\mathbf{f}^{\mathbf{o}_1}$ and $\mathbf{f}^{\mathbf{o}_2}$ at frame $i$, and $M^{\alpha}_{q}(\phi, \theta, \alpha)$ computes the quaternion orientation of the manifold coordinates $(\varphi(i), \theta(i))$ at time $i$ (see Appendix A for details). This representation is therefore able to retarget the camera trajectories with different targets at different positions while maintaining the same on-screen locations of targets (see Figure 8).

### 4.2. Vision-based extraction of trajectories

There are multiple estimators for camera parameters (also called match movers). Here we rely on the non-commercial Voodoo Camera Tracker software developed by Hannover University. The tool provides us with a mean to reconstruct a camera trajectory from a given sequence of images – typically extracted from a video file at a constant time rate. As displayed in Figure 3 the tool detects feature points in the scene (such as Harris corner detector [HS88] or SIFT points [Low99]). It then computes the correspondence between feature points in two succeeding images in order to obtain a tracking of feature points using techniques such as KLT tracking or SIFT matching. As a result, the tool estimates the camera trajectory as well as the camera parameters (see Figure 3). The user then needs to manually specify the 3D location of the target (or couple of targets) involved in the scene using an oriented cube primitive – an easy task since Voodoo provides a point cloud representation of the scene and manipulators to add geometries. When the targets are moving (e.g. actors walking), no points are reconstructed. The user then needs to estimate the motion of the targets and animate simple geometries accordingly (motions can be checked by overlapping the geometries on the video). The trajectories are then exported and converted to the appropriate representation ($G_f, G_1$ or $G_2$).

### 4.3. Mocap-based extraction of trajectories

Virtual camera systems (VCS) are devices developed in the virtual production industry to rehearse shots and edits in 3D environments before shooting in real environments. A virtual camera system encompasses a 6DOF tracked device (a rigid body representing the camera rig) together with a display on the rig to visualize the viewpoint from the rigid body's configuration in the virtual scene. In our approach, a plugin was written for Motion Builder (an Autodesk animation software) to extract the trajectories through using the VICON Tracker plugin. The reference targets are simply specified by the user (and their coordinates automatically exported with the trajectories).

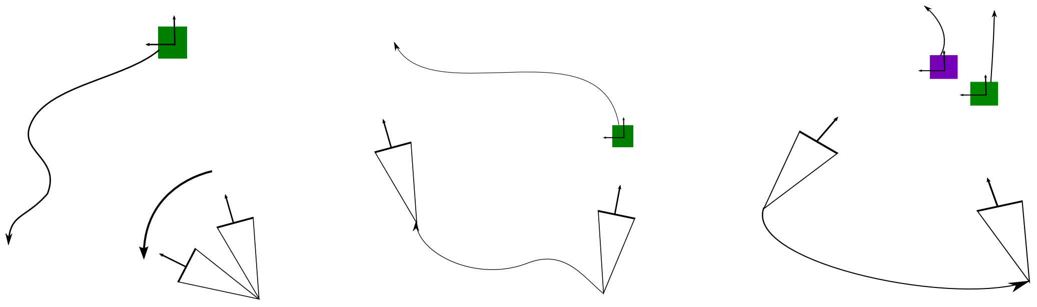
### 4.4. Trajectories and contextual information

While in theory, all trajectories could be seamlessly used in the motion graph, in practice, we propose to annotate trajectories with contextual information. Indeed, the conveyed meaning of a trajectory is often related to the nature of actions performed by the reference targets, or the intended communicative goal of the shot. We propose to re-use this contextual information when selecting appropriate trajectories (see Section 6). A filtering process is performed to only retain the trajectories matching the context of the new 3D scene (contextual information provided by the 3D engine, such as character actions or intended communicative goals). A contextual information is simply defined as a set of contexts. A context represents a lightweight annotation of the action performed by the reference targets of the scene (e.g. "A is_running", "A is_talking_to B", "A is_fighting_with B" when considering characters), or intended communicative goal of the camera along the camera trajectory ( such as "establishing A" or "establishing_relation A B"). Contextual information also holds the number and name of reference targets involved. Two contextual information match as soon as they share the same number of targets, and at least one action or intended communicative goal.

All trajectories annotated with their contextual information are then gathered in a database from which a camera motion graph will be constructed.

## 5. Constructing Camera Motion Graphs

The construction of a camera motion graph consists in sampling over time every pair of camera trajectories in order to



a) fixed camera motion b) single target motion c) multiple target motion

**Figure 2:** *Our different categories of camera motions.*

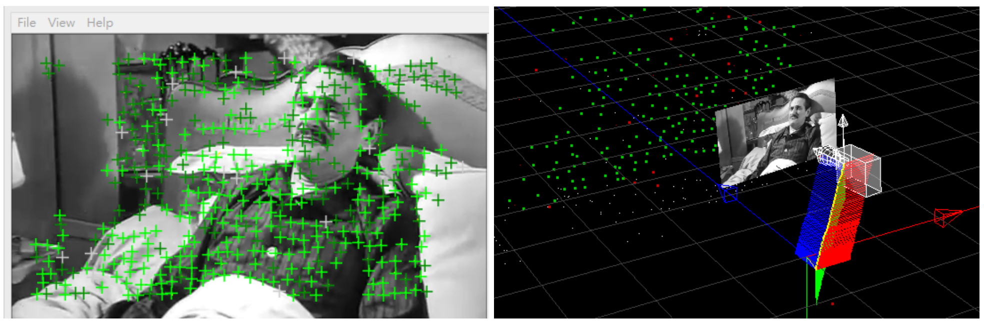


**Figure 3:** *Harris corner detector for feature points of a picture (on the left) that are used to reconstruct the camera trajectory from a sequence of images (on the right). The user then enriches the scene by placing target objects.*

evaluate the possibility and the quality of transitions between these trajectories. The purpose of the process is to build possible sequences of camera trajectories that satisfy continuity rules in cinematography (e.g. as defined in [Tho09]).

Two types of transitions are considered for a couple of camera configurations $\mathbf{q}_i$ and $\mathbf{q}_j$. Continuous transitions ($T^C$) represent generated pieces of trajectories that continuously link $\mathbf{q}_i$ and $\mathbf{q}_j$ together. Non-continuous transitions ($T^N$) represent cuts between configurations $\mathbf{q}_i$ and $\mathbf{q}_j$ (a sudden change in viewpoints). We also consider self-sampling which consists in evaluating the possible cut transitions between two camera configurations $\mathbf{q}_i$ and $\mathbf{q}_j$ of the same trajectory. Self-sampling is useful in case of short-timed occlusions of the targets, the camera switching to a further configuration in time on the same trajectory ($j > i$). Continuous transitions in self sampling was not considered, since the process will lead to shortcuts in the trajectory.

### 5.1. Non-continuous transitions $T^N$

A non-continuous transition (*i.e.* a cut) between two camera configurations $\mathbf{q}_i$ and $\mathbf{q}_j$ is considered possible when the cut satisfies cinematographic continuity rules. Two rules were considered: jump cuts and line of action.

Jump cuts occur when the angle between two cameras $\mathbf{q}_i$ and $\mathbf{q}_j$ with relation to a target $\mathbf{o}$ is lower than 30 degrees. This insufficient change in the content of the screen creates a visual discontinuity that should be avoided. However, if the difference between the on-screen projected height of a target in both shots is significant, the rule is overridden. The jump-cut cost $c_{JC}$ is expressed as:

$$c_{JC} = \begin{cases} 0: & \widehat{\mathbf{x}_{\mathbf{q}_i}\mathbf{x}_{\mathbf{o}}\mathbf{x}_{\mathbf{q}_j}} > \pi/6 \vee |p(\mathbf{q}_i, h_{\mathbf{o}}) - p(\mathbf{q}_j, h_{\mathbf{o}})| > t_h \\ c: & otherwise \end{cases}$$

where $c = \pi/6 - \widehat{\mathbf{x}_{\mathbf{q}_i}\mathbf{x}_{\mathbf{o}}\mathbf{x}_{\mathbf{q}_j}} + t_h - |p(\mathbf{q}_i, h_{\mathbf{o}}) - p(\mathbf{q}_j, h_{\mathbf{o}})|$ and $p(\mathbf{q}_i, h_{\mathbf{o}})$ represents on-screen projected height of the reference target $\mathbf{o}$ through camera $\mathbf{q}_i$, and $t_h$ represents a height threshold, here equal to 0.2 on a normalized unit screen.

The line of action rule states that the camera must not cross an imaginary line passing through two actors involved in a shot. The cost $c_{LA}$ of crossing a line of action establishes whether the two cameras $\mathbf{q}_i$ and $\mathbf{q}_j$ are the same side of the line $\mathbf{x}_{\mathbf{o}_1}\mathbf{x}_{\mathbf{o}_2}$:

$$c_{LA} = \begin{cases} 0 & : & \mathcal{P}^{\mathbf{o}_1\mathbf{o}_2}(\mathbf{x}_{\mathbf{q}_i}).\mathcal{P}^{\mathbf{o}_1\mathbf{o}_2}(\mathbf{x}_{\mathbf{q}_j}) < 0 \\ 1 & : & otherwise \end{cases}$$

where $\mathcal{P}^{\mathbf{o}_1\mathbf{o}_2}(\mathbf{x})$ evaluates whether point $\mathbf{x}$ is on the positive or negative side on the vertical plane $\mathcal{P}$ passing through targets $\mathbf{o}_1$ and $\mathbf{o}_2$. Cuts can also be performed between two camera configurations inside the same trajectory. In such case, we add an additional cost that prevents a cut to occur between a camera $\mathbf{q}_i$ and a camera $\mathbf{q}_j$ earlier in time, (i.e. such that $j < i$) to avoid replaying the trajectory.

### 5.2. Continuous transitions $T^C$

A continuous transition between two camera configurations $\mathbf{q}_i$ and $\mathbf{q}_j$ is considered possible when the cameras are close enough (cost $c_d$), do not have significantly different orientations (cost $c_\lambda$), and do not follow opposite directions (cost $c_{dir}$). Furthermore, transitions towards camera configurations too close to the end of their motion should be avoided (cost $c_{end}$). Costs are expressed as follows:

$$c_d = \begin{cases} 0 & : \quad |\mathbf{x}_{\mathbf{q}_i} - \mathbf{x}_{\mathbf{q}_j}| < 1 \\ |\mathbf{x}_{\mathbf{q}_i} - \mathbf{x}_{\mathbf{q}_j}| & : \quad otherwise \end{cases}$$

$$c_\lambda = \begin{cases} 0 & : \quad cos^{-1}(2(\mathbf{u}_{\mathbf{q}_i} \cdot \mathbf{u}_{\mathbf{q}_j})^2 - 1) < \pi/6 \\ 1 & : \quad otherwise \end{cases}$$

$$c_{dir} = \begin{cases} 0 & : \quad |\dot{\mathbf{x}}_{\mathbf{q}_i} - \dot{\mathbf{x}}_{\mathbf{q}_j}| < d_{dof} \\ 1 & : \quad otherwise \end{cases}$$

$$c_{end} = \begin{cases} 0 & : \quad e_{\mathcal{T}_2} - j < 30 \\ 1 & : \quad otherwise \end{cases}$$

where $d_{dof}$ represents a speed threshold under which the difference in camera speeds is acceptable, and $e_{\mathcal{T}_2}$ the last frame index of trajectory $\mathcal{T}_2$.

Figure 4 represents the cost of transitions (continuous and non-continuous) between two selected trajectories.

The sampling rate is empirically fixed to 0.2 seconds on each trajectory, which already generates a large collection of outgoing edges for each sample (an average of $n \times \mathcal{T}_d/0.2$ where $n$ is the number of trajectories in the motion graph and $\mathcal{T}_d$ is the average duration in seconds of the trajectories). Shorter sampling rates are possible but impact the cost of visibility testing in the real-time process (all outgoing nodes are tested for visibility when a transition is necessary). Longer sampling rates reduce the reactivity of the camera on sudden occlusions or changes in context.

## 6. Using Camera Motion Graphs

Once the camera motion graph is constructed in an off-line process, any real-time animation system that provides the coordinates of targets in real-time as well as contextual information can rely on the motion graph to compute realistic camera motions. When applying the camera motion graph to a new scene, the steps are the following. The system is first initialized by (i) accessing the current contextual information which also contains the reference targets involved,

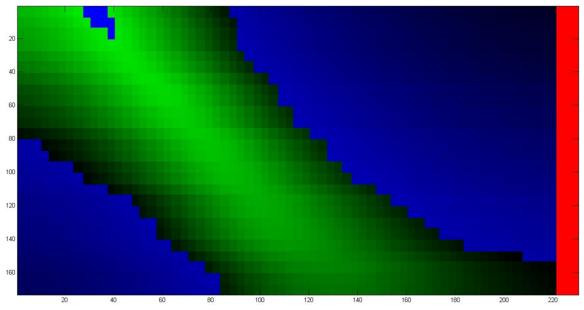


**Figure 4:** *Cost of transitions between two selected trajectories sampled at every frame. The horizontal axis represents the destination trajectory and the vertical axis the origin trajectory. The red area represents transitions that are not allowed (typically transitions towards configurations too close to the end of the destination trajectory). The blue areas are the cut transitions and the green areas the continuous transitions. The brighter the color, the lower the cost of the transition.*

(ii) filtering the camera paths compatible with the current contextual information (iii) selecting a the first camera path among remaining compatible candidates for which the visibility of target(s) in the first frame is ensured, and (iv) positioning the camera in the environment according to the selected camera path.

At each frame, the process then consists in testing whether a transition is necessary. When no transitions are necessary, the camera follows the current path. A transition is necessary whenever a visibility check fails on the targets (see Section 6.2), when the context has changed and is incompatible with the current path (see Section 6.1), or when the camera reaches the end of a path.

When a transition is necessary, a visibility check is performed on the reachable camera paths, *i.e.* on all the outgoing edges from the current node in the motion graph. All reachable paths have been prior ranked against the quality of their transition (performed in the construction of the motion graph), and are then filtered at run-time for compatibility in their context. With continuous transitions $T_C$, the first frame along the continuous transition is evaluated for visibility. With cut transitions $T_N$, the first frame on the destination trajectory is evaluated for visibility. The first transition ensuring visibility of targets in the ranked and filtered list is selected and taken, thereby performing a cut, or starting a continuous transition motion.

### 6.1. Change in contextual information

As a change in the contextual information of the environment occurs, a compatibility check with the context of the current camera path is performed. If incompatible, a transition is necessary. A change in the contextual information can be mean a change in the reference targets (switching from one target to another), a change in the actions performed or a change in the communicative goal.

The filtering over possible transitions simply consists in comparing the keywords and number of reference targets of the new context with those of the destination trajectories, and retaining the compatible ones.

### 6.2. Visibility of targets

A key issue to address in re-using existing camera paths in new 3D environments is ensuring visibility of reference targets. We propose the following approach.

At each frame, visibility of reference targets can be evaluated using a simple ray-casting technique along the current camera path. Yet, in order to avoid frequent transitions between trajectories due to partial or short-timed occlusions, we introduce *thick paths* on which we perform more evolved visibility tests, encompassed the evaluation of short term occlusions and partial visibility.

A thick path is a geometric shape placed around the immediate future of the camera path (see figure 5), and decomposed into regular cubes. A visibility check is performed by casting rays from the reference targets towards the cubes intersecting the camera path. If a cube on this path is occluded, all other cubes are evaluated for occlusion. We then rely on a local search process to find the closest unoccluded cube in the geometric shape and construct a path through this unoccluded cube. If no cube is visible, the visibility check fails.

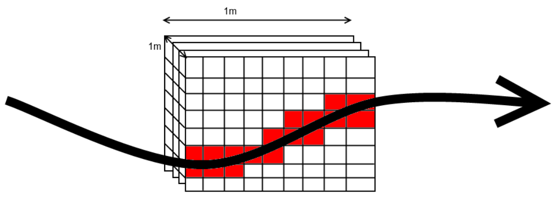


**Figure 5:** *A thick camera trajectory. The arrow represents the camera trajectory, the red cubes are the ones traversed by the camera. All traversed cubes are first evaluated for visibility, and in case of occlusion, the path is locally modified by finding a fully or partially unoccluded path through the set of cubes.*

We then improve the model by offering the possibility of specifying a visibility threshold $v_t \in [0;1]$. The ratio of visibility $v$ is then computed as the ratio of number of visible cubes intersecting the path, on the total number of cubes intersecting the path. The visibility check then fails when the ratio is below the threshold. We further extend the model by specifying a maximum duration of occlusion $d_t$ in millisecond. The visibility check then fails whenever the duration of occlusion along the cubes is above the threshold $d_t$. At each frame, the think path is moved ahead of the current camera position, and visibility is re-evaluated. Performance issues are discussed in Section 7. Figure 6 shows an example of visibility computation on 3 different trajectories.

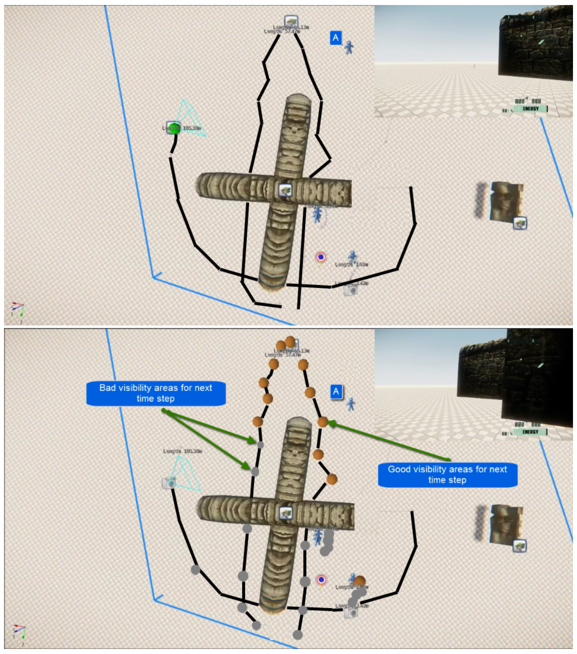

A
Bad visibility areas for next time step
Good visibility areas for next time step

**Figure 6:** *Topview of a sample scene illustrating the visibility computation. The upper picture shows the camera (green point on the left) and the target A (on the right) in an unoccluded situation (A is visible). Visibility is only performed along the current camera path (in black). The bottom picture shows the scene where A is no longer visible. All the nodes (reachable through outgoing edges representing cuts) are evaluated for visibility. Gray nodes represent viewpoints in which the target is occluded, and orange nodes viewpoints in which the target is visible.*

## 7. Results

We implemented and tested our approach on different environments using the *CryEngine*$^{TM}$ game engine to measure benefits and performances. Three criteria were retained:

- simplicity;
- adaptability;
- efficiency.

### 7.1. Simplicity in re-using data

The principle of our method is to import trajectories from real data into synthetic worlds. Just focusing on a single camera trajectory, Figure 7 shows an example of the imported trajectory, together with the original trajectory, illustrating the ease in reusing existing data. The simplicity is also illustrated by Figure 8 which demonstrates how for a camera, the same on-screen composition is maintained for different configurations of targets objects.

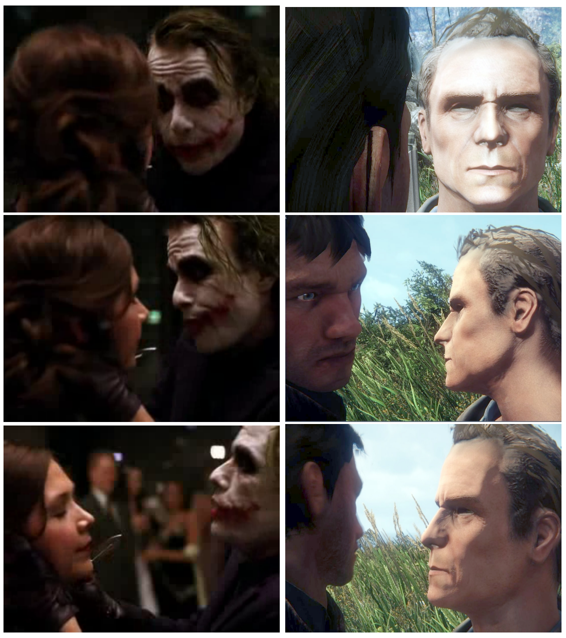


**Figure 7:** *Simplicity in reproducing a trajectory. In this example, the left images are the original pictures where the trajectory was extracted from. The right pictures represent the importation of the trajectory into our virtual world. Results in term of composition (position in are similar.*

## 7.2. Adaptability

Figure 9 shows an example of spatial and temporal adaptability. The feature presented here highlights the maximum duration of occlusion threshold ($d_t$).

## 7.3. Efficiency

Given that the process consists in replaying pieces of camera paths adapted to new environments, the main cost comes from the visibility tests. Visibility is computed at each frame. While a thorough testing is necessary, on the scenes we tested, an average of 500 rays were casted at each frame (this is including situations with occlusions and not). To test the scalability, the approach was pushed to 15000 rays per frame on a Dell with Intel Core I7 CPU 2.60GHz, a point at which the animation seriously lags. Over the different animations, the average time spent in our process is 16ms (including the

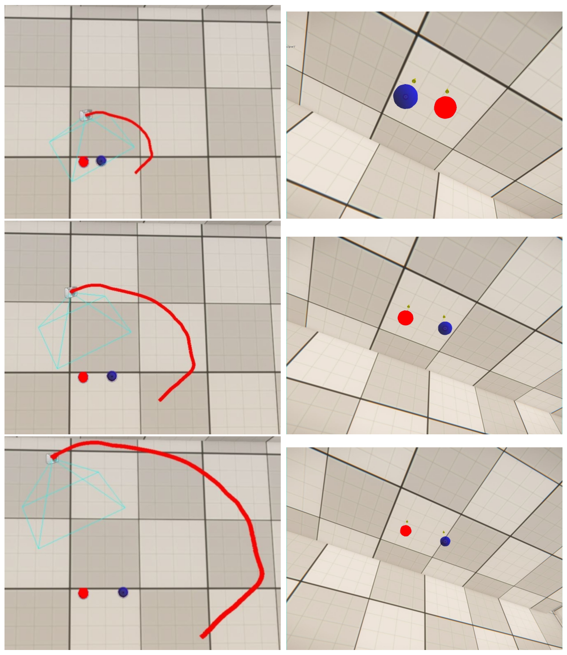


**Figure 8:** *Example of trajectory retargeting. In this example, a trajectory was imported into the scene. The reference targets for this trajectory are two targets represented by the red and blue points. Depending on the distance between the targets, the trajectory is adapted so that the composition in the synthetic shots matches the composition of the original shots.*

cost of visibility tests and computing a new path for the camera when in visually cluttered situations. In cluttered situations, a peak value of 32ms was reached.

Figure 10 and Figure 11 present examples of sequences generated with our camera motion graph process.

## 8. Limitations

Our camera motion graph is an efficient method to retarget real camera trajectories to virtual environments. It nevertheless presents some limitations. First, in building the motion graph, any transition that satisfies the cinematographic continuity rules is accepted. However the reality shows that choosing the right moment to cut, and the right shot to cut to is far more complex than conforming to basic continuity rules. Second, the retargeting of trajectories can lead to unaesthetic trajectories as soon as the paths of the reference targets in the real camera trajectory and in the virtual environment differ significantly. Currently, the contextual information prevents some of these issues, but better models are required that would account for both the trajectory of the targets and of the cameras.

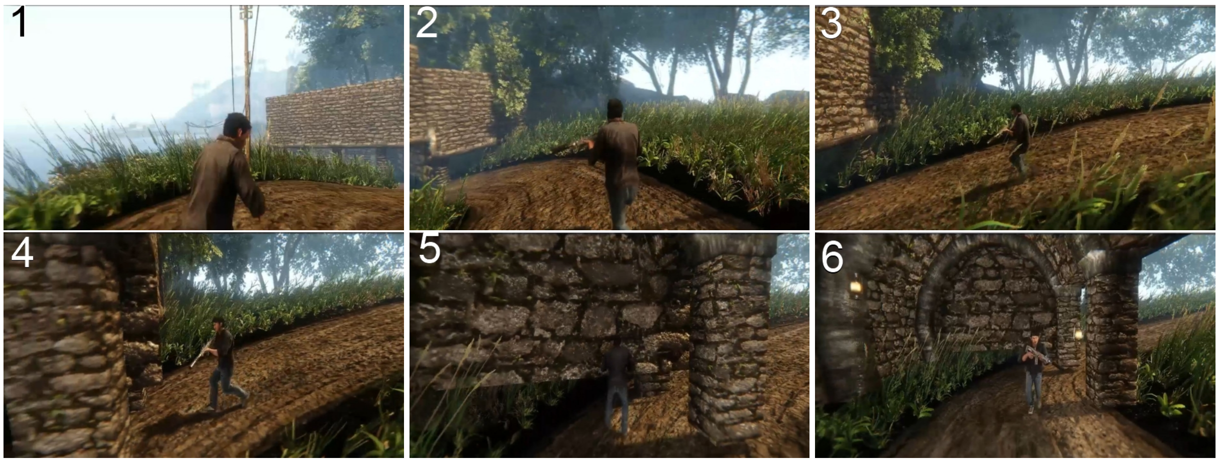


**Figure 10:** *Resulting shots from trajectories generated by our camera motion graph process, tracking a single target and cutting between trajectories when occlusion occurs (see shots 4 and 5).*

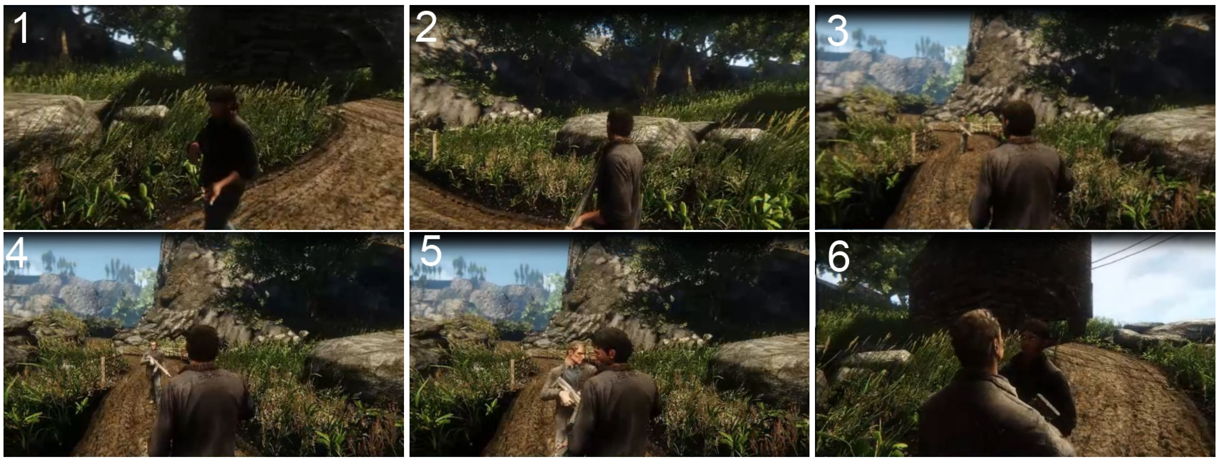


**Figure 11:** *Resulting shots from trajectories generated by our camera motion graph process, making a transition between a one target trajectory and a two target trajectory (between shots 2 and 3).*

## 9. Conclusion

Improving realism of camera control by the use of data extracted from real cinematography is a key challenge. In this paper we first presented a way to retarget different types of camera trajectories extracted from real footage. And we then proposed to organize camera trajectories in a camera motion graph inspired by character animation techniques. The trajectories are expressed in a normalized representation. The camera motion graph is then constructed by sampling pairs of camera trajectories for possible transitions. Finally at runtime, the camera motion graph is positioned according to new targets in a new 3D environment, and visibility is computed to ensure the proper viewing of reference targets. The approach provides us with a means to transit between different viewpoints while preserving the quality of shots and respecting cinematographic rules, while being simple and efficient.

## References

[CNO12] Christie M., Normand J.-M., Olivier P.: Occlusion-free camera control for multiple targets. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012), Eurographics Association, pp. 59–64. 1

[CON08] Christie M., Olivier P., Normand J.-M.: Camera control in computer graphics. In *Computer Graphics Forum* (2008), vol. 27, Wiley Online Library, pp. 2197–2218. 2

[HHS01] Halper N., Helbing R., Strothotte T.: A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. In *Computer Graphics Forum* (2001), vol. 20, Wiley Online Library, pp. 174–183. 1

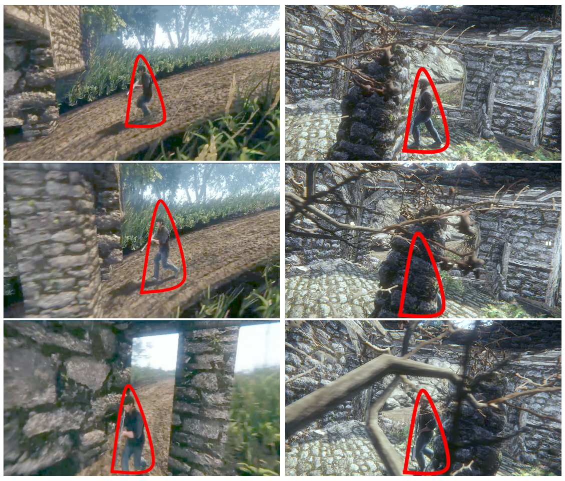


**Figure 9:** *These examples illustrate the duration of occlusion $d_t$ in visibility computation. On the left sequence duration of occlusion is set to* 0 *seconds, meaning that no occlusion is tolerated. This results in a cut towards a camera configuration further on the trajectory (in this example) when the character gets occluded. In the right sequence the camera path is not changed due to a duration of occlusion lower than the maximum duration of occlusion. Here the threshold is set to* 0.6 *seconds. For the right example, the occlusion lasts* 0.3 *seconds.*

[HS88] HARRIS C., STEPHENS M.: A combined corner and edge detector. In *Alvey vision conference* (1988), vol. 15, UK, p. 50. 5

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 473–482. 2, 3

[KRE*10] KURZ C., RITSCHEL T., EISEMANN E., THORMAHLEN T., SEIDEL H.-P.: Camera motion style transfer. In *Visual Media Production (CVMP), 2010 Conference on* (2010), IEEE, pp. 9–16. 2

[LC08] LI T.-Y., CHENG C.-C.: Real-time camera planning for navigation in virtual environments. In *Smart Graphics* (2008), Springer, pp. 118–129. 2

[LC12a] LINO C., CHRISTIE M.: Efficient composition for virtual camera control. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012), SCA '12, pp. 65–70. 2

[LC12b] LINO C., CHRISTIE M.: Efficient composition for virtual camera control. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012), Eurographics Association, pp. 65–70. 4, 11, 12

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 491–500. 3

[Low99] LOWE D. G.: Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on* (1999), vol. 2, Ieee, pp. 1150–1157. 5

[NO03] NIEUWENHUISEN D., OVERMARS M. H.: *Motion Planning for Camera Movements in Virtual Environments*. Tech. rep., Utrecht University, 2003. 1, 2

[OSTG09] OSKAM T., SUMNER R. W., THUEREY N., GROSS M.: Visibility transition planning for dynamic camera control. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), ACM, pp. 55–65. 1, 2

[SB04] SINGH K., BALAKRISHNAN R.: Visualizing 3d scenes using non-linear projections and data mining of previous camera movements. In *Proceedings of the 3rd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa* (2004), AFRIGRAPH '04, ACM, pp. 41–48. 2

[Tho09] THOMPSON R.: *Grammar of the Edit*. Focal Press, 2009. 2, 6

[VMGL12] VO C., MCKAY S., GARG N., LIEN J.-M.: Following a group of targets in large environments. In *Proceedings of the Fifth International Conference on Motion in Games* (2012), Bekris K., Kallmann M., (Eds.), Springer. Invited Paper. 2

[Zhe13] ZHENG C.: One-to-many: Example-based mesh animation synthesis. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2013), SCA '13, ACM, pp. 145–153. 2

## Appendix A: Manifold representation

The space of camera locations that satisfy the exact on-screen composition of two 3D targets can be defined by a 2D manifold surface (a subset of a spindle torus, see Figure 12). For any couple (φ, θ) on the 2D manifold, one can algebraically derive a camera position and a camera orientation that satisfy this exact composition [LC12b].

Let $A$ and $B$ be two distinct points in the 3D space representing the target points to frame. A first quaternion $q_\varphi$ is built as the rotation of angle $\varphi$ around the axis $\frac{\vec{BA}}{BA}$; and a second quaternion $q_\theta$ is built as the rotation of angle $\frac{\theta}{2}$ around an axis $t$ computed as

$$\vec{t} = \vec{z} \times \frac{\vec{BA}}{BA}$$

where $\vec{z}$ is a reference vector such that the plane $(A, \vec{AB}, \vec{v})$ contains all viewpoints $(\varphi, \theta)$ with $\varphi = 0$. $\vec{v}$ is computed as:

$$\vec{z} = \vec{AB} \times \vec{u} \text{ scaled to unit length}$$

$\vec{u}(0,0,1)$ being the world up vector.

Then the camera position $x$ corresponding to the couple $(\varphi, \theta)$ on the manifold surface is computed as:

$$x = A + q_\theta \cdot q_\varphi \cdot \vec{AB} \cdot \frac{\sin\left(\alpha + \frac{\theta}{2}\right)}{\sin\alpha}$$

$\alpha$ being the angle generating the manifold surface. The value of $alpha$ is computed as $angle(p^3_B, p^3_A)$, where $p^3_A(p_A.x/S_x, 1, p_A.y/S_y)$ and $p^3_B(p_B.x/S_x, 1, p_B.y/S_y)$ rely on $p_A, p_B$ respectively representing the projection of A and B on the screen, and $S_x = 1/tan(\gamma_x/2)$, $S_y = 1/tan(\gamma_y/2)$, where $\gamma_x$ and $\gamma_y$ respectively represent the horizontal and vertical aperture of the camera.

Given the camera position $x$, we can compute the camera orientation that ensures the composition.

We first build a quaternion $q_i$ which represents a first "default" composition of $A$ and $B$ (i.e. $y_A = y_B = 0$ and $x_A = -x_B$). $q_i$ is built using the basis $\overrightarrow{f_i}$, $\overrightarrow{r_i}$, and $\overrightarrow{u_i}$, where $\overrightarrow{u_i} = \frac{\overrightarrow{PB}}{|\overrightarrow{PB}|} \times \frac{\overrightarrow{PA}}{|\overrightarrow{PA}|}$ ; $\overrightarrow{f_i} = \left( \frac{\overrightarrow{PB}}{|\overrightarrow{PB}|} + \frac{\overrightarrow{PA}}{|\overrightarrow{PA}|} \right)$ scaled to unit length ; and $\overrightarrow{r_i} = \overrightarrow{f} \times \overrightarrow{u}$.

We now compute the rotation $q_c$ such that, when applied to $q_i$, points $A$ and $B$ are projected in the appropriate locations on the screen (*i.e.* $p_A$ and $p_B$).

$$q = q_i \cdot (q_c)^{-1} \qquad (1)$$

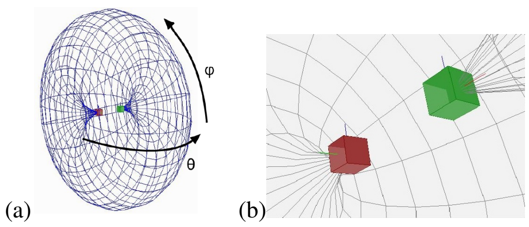


**Figure 12:** *(a) Given desired on-screen locations of two targets, the manifold representation defined by [LC12b] provides the surface of viewpoints for which the on-screen locations are exact. (b) a possible viewpoint is defined as a couple of angles* ($\varphi, \theta$) *on this surface.*