Camera Control with Motion Graphs

Clément Désoche

Research Master Thesis

Research Master Thesis

Camera Control with Motion Graphs

Author:
Clement Desoche

Supervisor:
Marc Christie
EPI MimeTIC

Abstract

This report presents Camera Motion Graphs, a new 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 those trajectories and (iii) transitions representing cuts between two 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 express all the camera trajectories in a normalized representation, making camera paths easily adaptable to new 3D environments. Specific metrics are then detailed to compute the best transitions between camera paths. Multiple results illustrate the simplicity of the technique, its adaptability to different 3D environments and its efficiency.

Contents

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. Typically, computer games rely increasingly on elements of style/genre from real movies in terms of camera placements, trajectories and edits. In the gaming industry, these issues are mostly addressed through collections of manually crafted camera animations.

The path towards a fully automated computation of cinematography is complex. Current contributions address the problem by procedurally generating camera trajectories. Using optimization-based 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: first, a number of generated paths are impossible to reproduce using real cameras even through complex devices such as robotic arms or Louma systems. And through many years of training, audience is now educated to certain types of camera trajectories. Second, there are specificities and subtleties on camera motions which are difficult to reproduce with computational models: variations in speed at the beginning and end of trajectories, or noise in motions due to 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 these existing paths to new 3D environments? One first needs to reproduce the proper framing of targets (i.e. on-screen locations, 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 3D environments. And Third, the visibility of targets must be evaluated and strategies must be developed to avoid occlusions.

Our goal in this report is to maintain the realism of camera paths by extracting them from real data (existing movies or motion captured data), and provide means to adapt these paths to new 3D environments. Directly inspired by the way character animation techniques strongly rely on motion captured information to ensure realism, 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 cuts representing 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 find possible transitions. The cost of a possible transition is defined using cinematographic continuity criteria.

This report describes quick state of the art of the camera control domain and then the three stages of 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 new 3D environments.

2 Related Work

Controlling a virtual camera is a complex problem. A survey on the different techniques has been proposed by Christie et al.[CON08]. The domain of camera control can divided into three main issues. First defining good shot, then determined good trajectories and finally performing good transition between the shots.

2.1 Viewpoint computation

The first question to be asked in camera control is how to determine a good viewpoint? As the main goal of camera control is to transfer information from the scene to the viewer, measuring the amount of information conveyed through the camera can be interesting. It is possible to define a measure called picture's entropy, by similarity with the one defined by Shannon[CT12]. Several kind of picture's entropy can therefore be defined. The will refer to the first as geometric entropy, because its computation relies on the scene geometry and the projection of this geometry in the picture. The second will be called motion entropy, a measurement used to compare the movement perceived in the picture sequence, with the real movement in the scene. And the third will be called cinematographic entropy as it is based on shot composition's rules used in cinematography

2.1.1 Geometrical entropy

Vàsquez et al.[VFSH01] propose to compute an optimal viewpoint with a geometrical measure: the viewpoint's entropy. The principle is to estimate the area covered in the picture by the projection of the object. The candidate viewpoints are sampled on a sphere S around the target object and evaluated for their entropy. The entropy is computed with the following formula:

$$\sum_{i=0}^{N_t} \frac{A_i}{A_t} \log \frac{A_i}{A_t}$$

N_t is the number of faces in the scene, A_i is the projected area of face i and A_t the total area of the sphere S. The relation between A_i and A_t represents the visibility between the face i with respect to a point p. This relation between the object's projected area and the total area of the sphere is proportional to the cosines between the area's normal and the viewpoint direction and inversely proportional to the square of distance between the viewpoint and the object. The idea to use geometric data to compute viewpoint's quality has been extended by Vieira et al. [VBP+09]. The authors propose to take into account information directly linked to the 3D mesh of the object. Among those information there is the mean curvature, visible 3D surface, foreground alignment and silhouette complexity. Using those descriptors on existing objects, the authors propose a statistical learning method to determine the best viewpoints for new objects, by studying manually authored best viewpoints on a predefined set of objects.

2.1.2 Motion entropy

Geometrical entropy is an efficient method to assess a static viewpoint quality. Still in virtual cinematography most of the scenes are dynamic. In those scenes, the amount of information is directly linked to the movement. Assal et al. [AWCO10] proposed a method to measure the quality of a viewpoint related to the quantity of movement.

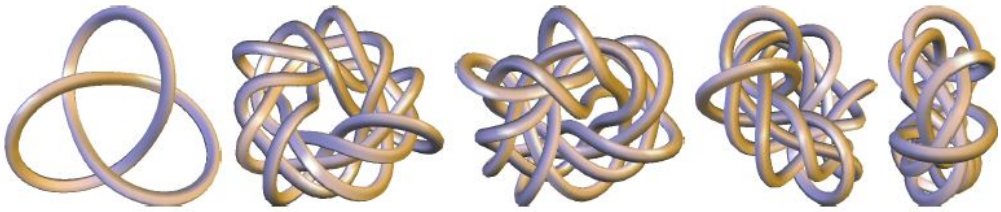


Figure 1: Learning on the first knot, Vieira et al. [VBP+09] compute the other viewpoints (classified from best to worst, left to right) on a more complex mesh.

This measure is used to compute the best viewpoint in real time. The scene is rendered by cameras in different positions around the scene. The authors determine the correlation between the movement in the scene and the movement perceived from different viewpoints. The viewpoint where the motion on the screen is most correlated to the motion in the scene transmits most of the scene information.

The correlation measure is computed using the CCA method (Canonical Correlation Analysis). This method computes the dependency between two sets of random variables, the authors apply this method on time windows of 40 frames. Unfortunately the CCA uses matrices are too big for this time window. This problem is solved using kernel matrices that simplify the problem. The video output on the different viewpoints are represented by k sets $x_{i_n}^1$, each one corresponding to one picture. The movement is represented by n vectors $y_{i_n}^1$, linked to the skeleton's articulation. The CCA is applied on pairs of vector x_i^k and y_i. The CCA outputs two results: the canonical correlation between the k viewpoints and the number of canonical correlations superior to a given threshold. Those values measure the viewpoint quality with respect to the motion taking place in the scene and the motion displayed on the screen.

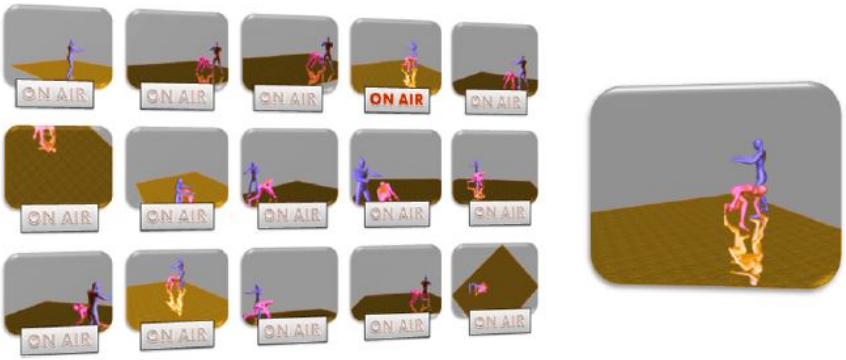


Figure 2: Among all the shots, the method of Assa et al. [AWCO10] chooses the one where the movement are the best represented

The method finds experimental results close to practice in cinematography, since portraying motion is a choice often made by film directors [Mas65]. When the input data are composed of video flows, the correlation are computed between the flows. However the method tends to generate close shot and extreme close shots more often than other types of shots.

2.1.3 Cinematographic entropy

The movement entropy presented in the previous section allows a better estimation of the viewpoint quality with dynamic scenes but suffer from several limitations. First it limits the expressiveness of the user, he can't choose a style to apply on the different generated shots. As said before, it favours a certain type of shot and requires a knowledge on the movement in the scene. In order to address those limitations, a third measure can be defined. The idea behind this measure is to take advantage of the cinematographic rules empirically defined by film directors through cinema history. We refer to this measure as cinematographic entropy.

Burelli et al.[BDGER08] propose a method expresses the cinematographic properties as constraints to be satisfied. Then a search among viewpoints is launched to find the one that best satisfies the constraints. The properties link the scene objects to the final picture (for instance: the position and size of the object in the final picture, or the view's angle of the object). The properties can also constrain camera parameters (e.g. be out of the wall). The process is performed in two steps.

At step 1, the properties are used to compute volumes of possible camera positions, without taking occlusion into account as presented in [CN05]. At step 2 an optimization method (PSO: Particles Swarm Optimization) is run inside the volumes of possible camera positions. For this purpose, an agent population is initialized randomly (position and speed) in the search space, then the population's speed is iteratively changed, to make it tend to both the best position that the agent has encountered and the best solution of the entire population. In the problem of VCC (Virtual Camera Composition) the search space consists of 7 dimensions (3 position, 3 orientation, and 1 field of view). The position is constrained by the volume. The evaluation function used in the optimisation process using properties on the object and the occlusion.

2.2 Trajectory planning

Now that we defined how to find a good viewpoint to look at a 3D scene, we are going to present a representative subset of techniques that create continuous sequences of viewpoints (camera paths). The first approach we present uses a sequence of good viewpoints to explore the scene. The second locally plans the trajectory in the environment around a target to follow. And the third plans the trajectory in the global environment of the scene taking into account a target that must stay visible.

2.2.1 Viewpoint sequence

The first solution we present consists in using a sequence of good viewpoints and create a continuous sequence. In [VFSH01], we notice that the succession between the different viewpoints seems to be really abrupt see Figure 4. This method explores the scene from a random position. Then the method chooses between 3 directions to change the camera position. The chosen direction is the one maximizing the entropy or the one that allows to see new faces of the model or by default the farthest from the start position. All three entropies could be used in this kind of example.

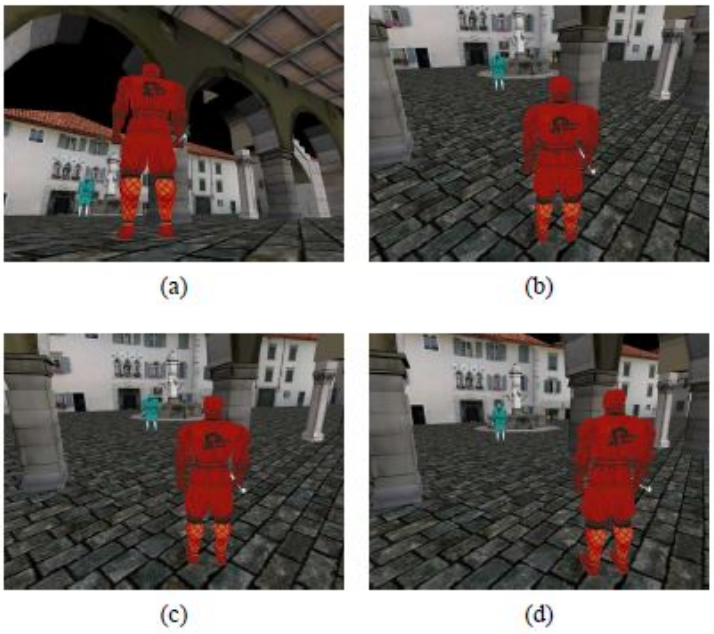


Figure 3: The desired shot composition here is a shot of the blue character over the shoulder shot of the red character. The (a) shot is obtained by an exhaustive search in a grid, the other shots are obtained by an execution of the method described in [BDGER08]

2.2.2 Local planing

The main issue with the previous method is that it computes unnatural camera trajectories and that it does not take into account dynamic occlusion. Another type of technique relies on trajectory planning, from the robotics field. For instance the article of Nieuwenhuisen and Overmars[NO03] or Li et al.[LC08] propose a trajectory planning method with occlusion avoidance, using a probabilistic road map.

A simple solution for third person cameras (i.e. cameras filming a character from an outside perspective) would be to fix the camera behind the character. But this technique produces undesirable occlusions, if for instance the character has his back on a wall. In their approach, Li et al.[LC08] wanted to solve strong constraints (no occlusion) and weak constraints (preferred camera position). The camera in this method is placed behind the character as a free-flying object, modelled with 6 degrees of freedom that have to be computed at each frame. To reduce the computational cost, the camera is always oriented towards the character, removing 3 degrees of freedom. The road map is build with the Lazy Roadmap [BK00] algorithm, and is represented as a graph with nodes and arcs around the character, nodes being the camera positions and arcs the transitions between the positions. A first map is built, then the algorithm tries to find an occlusion-free path in the map. If it fails, the bad positions are deleted and new one are added. Once the path is found, the algorithm searches for the path that minimizes a cost function during a limited period of time. The cost function are defined by cinematographic criteria.

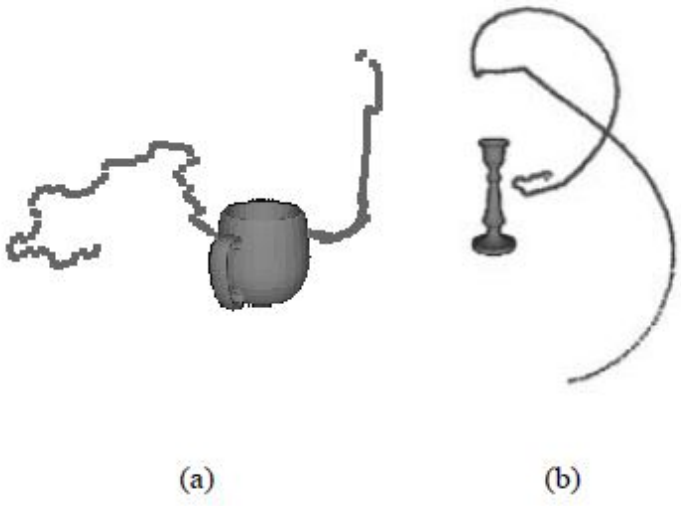


Figure 4: A trajectory obtained by using a succession of good shots with the method presented in [VFSH01].

2.2.3 Global planing

In the method described in [LC08], the authors planned the trajectory locally, i.e. only takes into account the environment close to the character. But this method fails when the character suddenly turns behind a wall. To correct this, planning techniques with a global knowledge of the environment are usefull. In the following we present an approache representative of global planning techniques [OSTG09].

This method consists in pre-computing a road map based on visibility in the whole scene. The scene is divided with a regular grid, the cells that contain 3D objects are eliminated, then each cell is filled with a sphere of maximal size. The intersections of the spheres are portals through which the camera can navigate. Some connections are then eliminated between pairs of spheres using a Monte Carlo ray casting technique to evaluate visibility. The portals become nodes in the road map and the transitions between portals are the arcs.

After this step, the built road map will be used for real-time planning. This process has two steps [HNR68]. First find a path in the map: the algorithm used is A*, using weights on the arc proportional to the pre-computed visibility. The second step will refine the path created in step 1, using GPU techniques. It computes occlusion maps for different portals that can be partially occluded. The process is real-time because the number of portals is low thanks to the pre-computation process. After those two steps, the path is still not fully determined: free space is available between the different portals. This free space is used to smooth the trajectory as an iterative process. If it produces occlusions the refinement process can be used again.

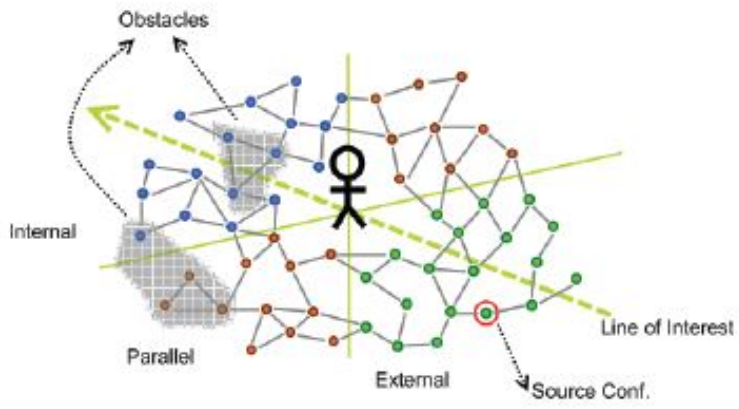


Figure 5: A road map computed around an avatar and used to compute the camera trajectories presented in [LC08].

2.3 Montage

2.3.1 Constrained cut

In the article [LC08], finding a continuous path without occlusion is not always possible. In this case, the camera changes by performing a cut. To allow those cuts, virtual links are introduced in the road map between two non-neighbour nodes. Some rules have to be followed to create the virtual link: the two extremities of the link have to be occlusion free, the end node has to be as far as possible from the begin node to avoid falling again in occlusion and to respect the line of interest rule. With this rule (see Figure 5) an imaginary line is placed along the direction of the character. The rule states that if the shot takes place on a side of the line, the shot following the cut has to be on the same side of the line. The map is divided into six zone defined by their view angle. The problem with this method is that the cut is only done if the visibility of the target is threatened and there is no continuous node to keep it. This technique is not suitable to produce a realistic sequence in virtual cinematography, as the visibility is not the only factor that can lead to a cut.

2.3.2 Measured helped cut

More recently, Assa et al. [AWCO10] propose to add a measure on the cut and some rules. The best shot is selected in real time and used until a erosion measure has reached a threshold. This erosion measure is computed using the accumulation view erosion signal (AVES) on a time window of 90 pictures. This erosion measure is computed with the actual quality of the current picture and the maximum quality of all the pictures that has been displayed before. The worse the current is the more the erosion raises. The new view is determined as follows: the view that induces jump cut (a cut between two similar viewpoint) and those who break the line of interest rule are eliminated. Then the one that are the less correlated to the motions are eliminated. Then the final candidates are ranked by the canonical correlation above a threshold and the number of pixels taken by the face of the actor.

This method allows more frequent cuts and are independent from the visibility issue. The

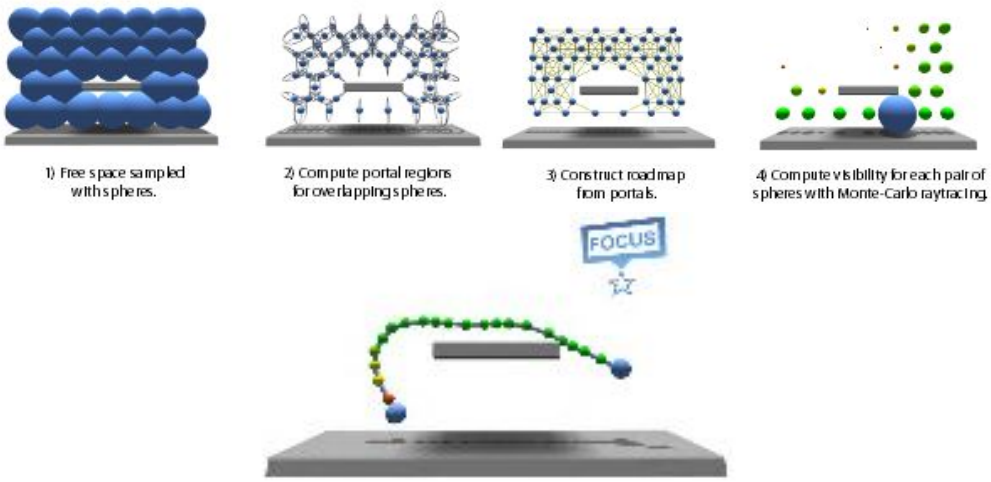


Figure 6: Road map computed in the environment of a 3D scene following the method of [OSTG09].

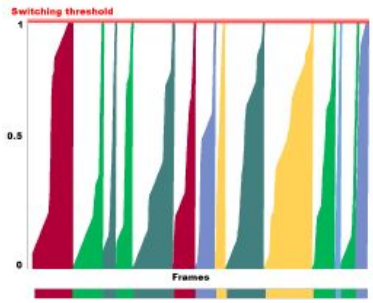


Figure 7: Evolution of the erosion measure presented in [AWCO10]

method also adds a new cinematographic rule: the jump cut.

2.3.3 Using cinematographic rules

The previous technique improves the cut between shots, but it only partially relies on cinematographic rules. The idea of He etal. [HCS96] is to fully base the cut on cinematographic principles, created by directors all along the cinema history (see [Tho09]). But such rules are difficult to encode in a computer understandable language.

The method developed in [HCS96] and called Virtual Cinematographer (VC), is designed to be used as a helper for creating a synthetic movie. VC is composed of two parts: cameras (that encode the camera positions) and idioms that formulate the cut rules. Sixteen cameras have been implemented. The module uses the actor position to automatically position the camera.

Each idiom (a stereotypical way of filming) is in charge of the expertise of a certain type of scene (an actor walking alone, two actor talking etc.). An idiom is in charge of choosing the shot, and performing the cut between shots. Idioms are also able to recognize when the situation no longer belongs to their field of expertise. The idioms are implemented as finite state machines. Each state

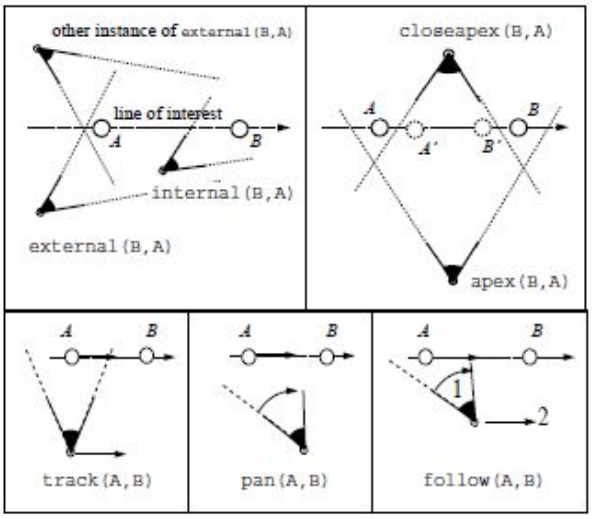


Figure 8: Example of typical shot used in the cinema [HCS96]

is associated with a camera module and condition to go from a cut to the other. For instance the idiom for two actor talking is composed of four states, two states the filming a character speaking and two other states for reaction shots. When a character is speaking, the corresponding shot is displayed, and cuts are performed from time to time towards the reaction shot.

The idiom for a 3 people talking is implemented as a parent of the previous one. It has four states, the first is a shot that establishes the situation the second uses the idiom of two people talking and the two last are reaction shots of the third actor. The idiom uses an exception system to take back the control of the child idiom.

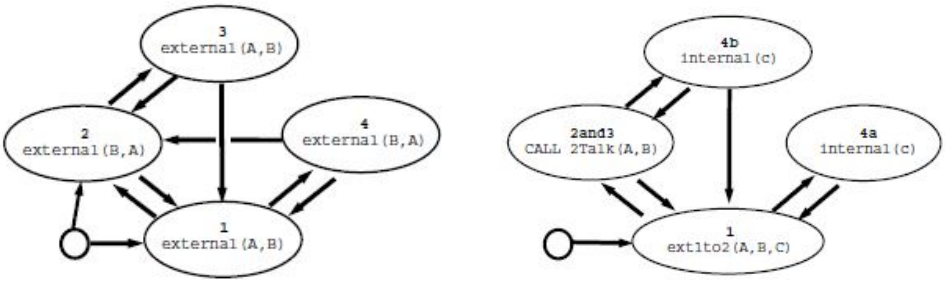


Figure 9: The idiom representing a conversation between two and three actors [HCS96]

2.4 Using real data

In the previous part, we have examined different techniques related to camera control. We have seen that the majority are using procedural algorithms to compute the camera parameters. This implies that the given results may seem unnatural they lack a human touch. Our idea is to compensate for this lack by using real data. In this part we are going to see how real data can be used to enhance

camera trajectories. We will also present how real data can be used and adapted tp create realistic human motions, a principle that we propose to re-apply for cameras.

2.4.1 Enhancing camera trajectories

In the article [KRE+10], the authors proposed a first attempt to use real data to enhance procedural generated trajectories. This method acts as a style transfer from real camera trajectory to virtual one. This process takes three steps. The first step is the acquisition phase, where the goal is to collect real camera trajectories and gather them in a common framework. The trajectories are represented as a succession of camera positions. As the trajectories have a limited time frame they can be stretched out by different techniques. The second step is the learning phase, each trajectory is split into two elements: the base and the detail. The base is computed by filtering the trajectory with a Taubin filter. The details is the difference between the base and the trajectory. The details is then converted in the frequency domain with a Gabor transform to get a deformation spectrum. Finally the key of the trajectory is the the derivative of the base. The third step is the modifying step: the user has a procedurally generated trajectory. In order to modify it, this step sends the derivative of this trajectory to compare it to the different keys of the base. Each key is associated to a deformation spectrum. The deformation spectrum for the trajectory is computed as a weighted sum of all the spectrum of the base with weight being a distance between the requested key and the key in the base.

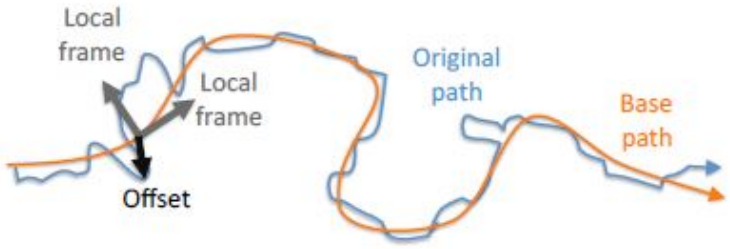


Figure 10: The trajectory decomposition proposed by [KRE+10], the original trajectory in blue decomposed in the based in orange and the detail being the difference between the two previous path.

2.4.2 Use of real data for human animation

The motion graph technique is used in the character animation domain. It is used to create human motion as described in [KGP02, LCR+02, GSKJ08] or for mesh deformation such as proposed by [Zhe13]. It aims, in the case of character motion, at animating virtual humans with data recorded by motion capture devices, in order to make the animation more realistic. A motion graph is a graph which nodes represent character poses and the arcs represent transition between those poses. The motion graph is computed from a set of motion clips, the clips are sampled in smaller ones and the transition are computed between each node. To compute those transition a distance metrics is used to determined if two poses are close or not. In the case of character animation the poses are represented by a set of angles associated with the joint of the character's skeleton, and the distance

metric used those parameters.

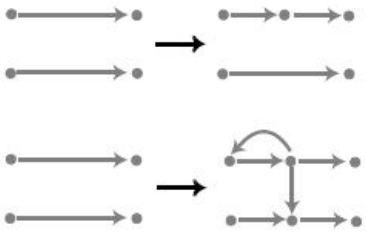


Figure 11: A motion graph construction from two clips. The clips are split into smaller ones and transitions are computed between the nodes [KGP02]

The motion graph is then used to synthesize the motion of the character at each time t the character poses is a node of the graphs and the next poses can be computed using the following node of the graph and the situation of the character in the 3D scene. The character can go smoothly from walking to running, crawling under an obstacle, the possibility of motion is only limited by the clips in the database.

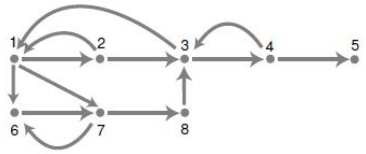


Figure 12: An example of motion graph[KGP02]

3 Our approach

As shown in Section 2, the vast majority of camera control techniques use a procedural approach. The main issue with that kind of procedure is that it creates trajectories and cuts that lack realism. Some approaches have already been proposed in order to use real data to enhance procedurally generated trajectories. Our idea is to go further and use the entire real trajectories to control the camera. We have shown that in other domain real data is successfully used to control 3D entities and we propose to extend those techniques to the problem of camera control. We name our structure the "camera motion graph" as it is a motion graph applied to camera trajectories. One of the main difference between the use of motion graphs between the domain of character animation and the camera control domain is that in the first one the goal is to put the character in the right position at a given time whatever has happened before, whether than in camera control the goal is more on the succession of camera poses than on positioning the camera at a given place at a given moment. To put it differently, the choice of the trajectory have to make sense with respect to the action played by the actor. Our camera motion graph is a directed graph. 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. 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 can be added 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.

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

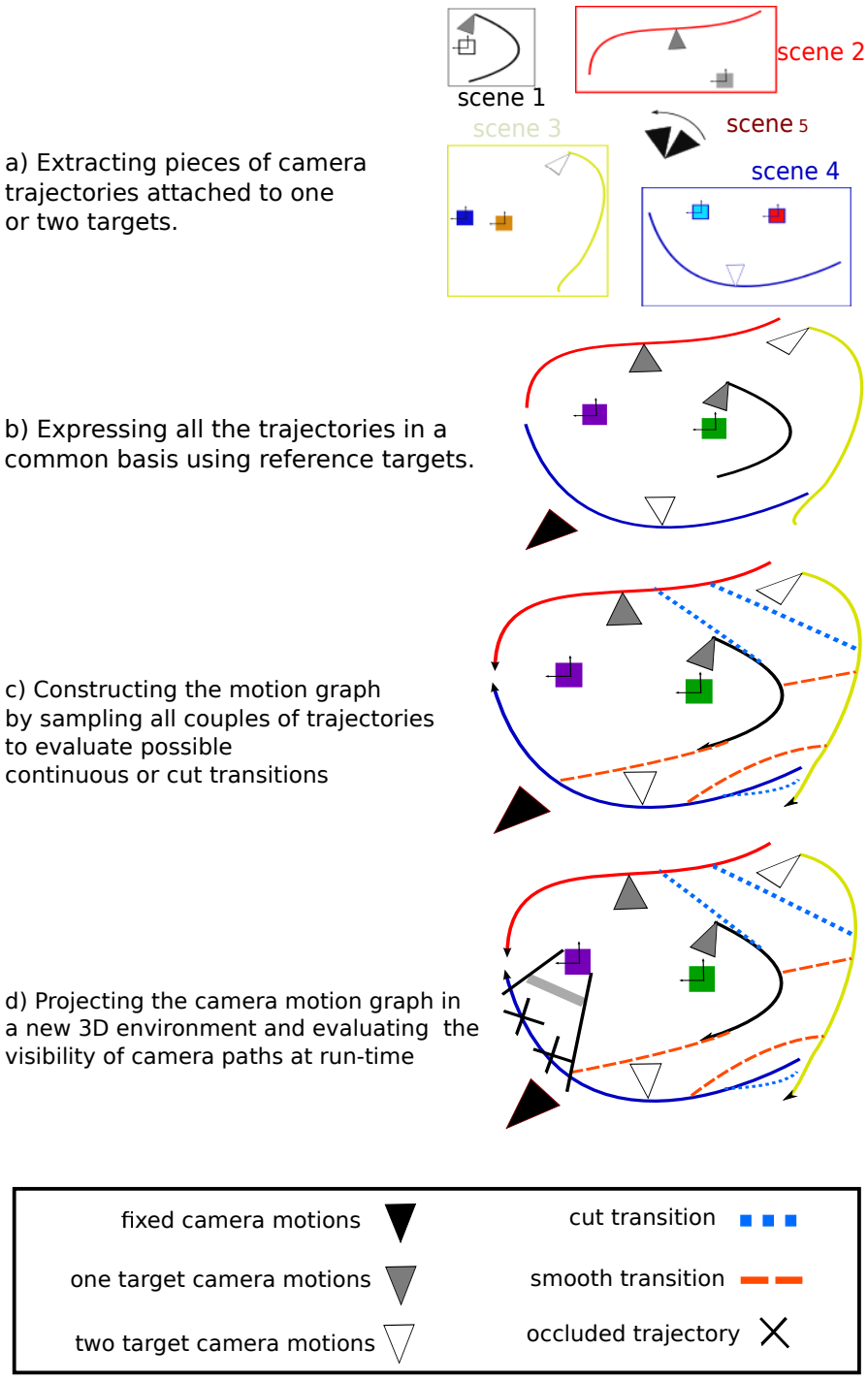


Figure 13: Process overview.

4 Modelling the problem and extracting the real data

4.1 Representation of camera trajectories

Camera trajectories extracted from tracking tools or mocap software are generally 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 suitable for creating a camera motion graph, a better representation is required.

Let us first define some notations. Let $\mathbf{q}$ be a camera configuration. A camera configuration is defined by its Cartesian position $\mathbf{x_q}$, its quaternion orientation $\mathbf{u_q}$, its field of view $\gamma_\mathbf{q}$, and its 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. Let $\mathbf{o}$ be a target in the scene defined by its Cartesian position $\mathbf{x_o}$ it's quaternion orientation $\mathbf{u_o}$ and it's height $h_\mathbf{o}$. A framing configuration $\mathbf{f}_\mathbf{q}^\mathbf{o}$ is defined by a 2D position representing the on-screen projection of a target object $\mathbf{o}$ through a camera $\mathbf{q}$.

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). The second (denoted G_1) gathers camera trajectories which mainly focus on a single target. Example trajectories are turning around a character or performing a travelling while following a character. And the third group (G_2) gathers camera trajectories defined over two or more targets. Example trajectories are establishing camera motions, circular motions around the characters, or crane motions. All three groups are represented in Figure 15.

We then consider that the targets will be the basis on which to create our normalized representation in which to express the camera trajectories. We refer to these targets as *reference targets*. While the task of expressing trajectories in the basis of a single reference target is trivial, the problem is more complex with two reference targets (see Section 4.1.3). The spatial configuration of these targets are indeed specific to each extracted trajectory.

Furthermore, rather than storing a sequence of camera orientations $\mathbf{u_q}$ that frame the reference target(s), we store a sequence of framing configurations $\mathbf{f}_i$ for each target (ie where the reference target is projected on the screen). The benefit of this representation is to enable the replay of camera motions that will maintain the framing positions of new targets on the screeen 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 motion of the new targets does not correspond to the motion 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 15(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 $\mathbf{q}_i$ in the local basis $\mathbf{o}_0$ of the target at it's first frame ($\mathbf{o}_0$ representing the target configuration at frame 0). The camera's orientation is expressed using a framing configuration $\mathbf{f}_\mathbf{q}^\mathbf{o}$ (projected position of target $\mathbf{o}$ on the screen). Field of view and depth of field are unchanged. In order to address the scaling issue (dealing with targets which size is different from the reference targets), 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 position $\mathbf{x}_q$ of the camera so that the projected height of the target is the same.

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

$$\begin{cases} \mathbf{x}_{\mathbf{q}'} &= \mathbf{x}_{\mathbf{o}'}.\mathbf{u}_{\mathbf{o}'}.\mathbf{u}_{\mathbf{o}_0}.h_{\mathbf{o}'}/h_\mathbf{o}.\mathbf{x}_{\mathbf{o}_0} \\ \mathbf{u}'_\mathbf{q} &= \mathbf{u}_\mathbf{f} \\ \gamma'_\mathbf{q} &= \gamma_\mathbf{q} \\ \rho'_\mathbf{q} &= \rho_\mathbf{q} \end{cases}$$

where $u_\mathbf{f}$ represents the camera orientation computed from the framing configuration $\mathbf{f}_\mathbf{q}^\mathbf{o}$.

4.1.2 Single target motions G_1

In the group of trajectories G_1 (see Figure 15(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. The camera orientation is expressed with a framing configuration $\mathbf{f}_\mathbf{q}^\mathbf{o}$ defined at frame i on target $\mathbf{o}$. Field of view and depth of field are not modified. The camera configuration is therefore defined as $[\phi, \theta, d, \mathbf{f}_\mathbf{q}^\mathbf{o}, \gamma, \rho]^T$ where ϕ, θ, 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 as:

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

where $S(\phi, \theta, d.h'_\mathbf{o}/h_\mathbf{o})$ computes the transformation from spherical coordinates (ϕ, θ, d) to Cartesian coordinates, and $\mathbf{u}_\mathbf{f}$ represents the camera orientation computed from the framing configuration $\mathbf{f}_\mathbf{q}^\mathbf{o}$.

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 and its two reference targets. 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.* [LC12]. The manifold coordinate system was initially defined to perform efficient tracking

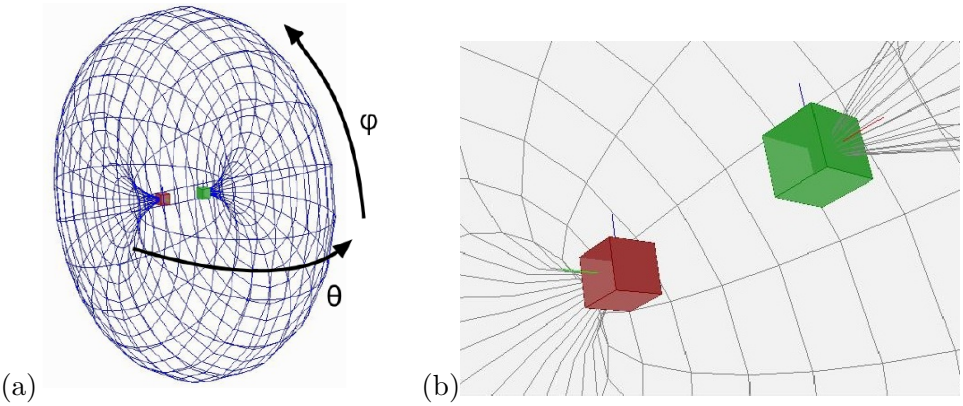
of two or three dynamic targets while maintaining a fixed specified composition on the screen (see Figure 14). The principle, given the desired on-screen location of two targets and the 3D positions of these targets in the scene, is to compute a manifold surface on which each and every point ensures the desired on-screen locations. We propose to re-use this representation by expressing our camera trajectories for two targets on manifold surfaces.

In the group G_2, 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 specifed using two compositon configurations $\mathbf{f}_{\mathbf{q}}^{\mathbf{o}_1}$ and $\mathbf{f}_{\mathbf{q}}^{\mathbf{o}_2}$ (see Figure 15(c)). The configuration is thus defined by the following parameters $[\varphi, \theta, \alpha, \mathbf{f}_{\mathbf{q}}^{\mathbf{o}_1}(t), \mathbf{f}_{\mathbf{q}}^{\mathbf{o}_2}(t), \gamma, \rho]^T$ where φ, θ, α 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 as:

$$\begin{cases} \mathbf{x}'_{\mathbf{q}} &= Mp(\phi, \theta) \\ \mathbf{u}'_{\mathbf{q}} &= Mr(\phi, \theta) \\ \gamma'_{\mathbf{q}} &= \gamma_{\mathbf{q}} \\ \rho'_{\mathbf{q}} &= \rho_{\mathbf{q}} \end{cases}$$

where $Mp(\phi, \theta)$ provides the Cartesian expression of the manifold coordinates (ϕ, θ) associated with desired compositon $\mathbf{f}_{\mathbf{q}}^{\mathbf{o}_1}$ and $\mathbf{f}_{\mathbf{q}}^{\mathbf{o}_2}$, and $Mr(\phi, \theta)$ computes the quaternion orientation of the manifold coordinates (ϕ, θ). This representation is therefore able adapt the camera trajectories according to the distance between the targets. And it maintains the same on-screen locations of targets.

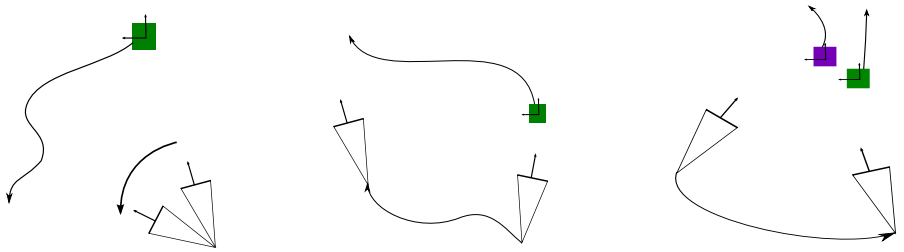


(a) (b)

Figure 14: (a) Given desired on-screen locations of two targets, the manifold representation defined by [LC12] provides the surface of viewpoints for which the on-screen locations are exact. (b) A possible viewpoint for a couple of (θ, ϕ) values.

4.2 Trajectories context

In practice, the geometrical representation of the trajectories expressed in the basis of the target objects is not enough to provide a qualitative animation. The meaning of a trajectory is often attached to the nature of actions performed by the target objects. Therefore we propose to extend



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

Figure 15: Our different categories of camera motions.

the geometrical representation with the context in which the camera trajectory was initially designed. A filtering process is then performed in real time to only retain the trajectories matching the context of the new 3D scene. A context is defined as a set of keywords. A keyword represents a lightweight annotation of the action performed by the target object of the scene (eg. running, talking, fighting when considering characters), or communicative intention of the camera along the camera trajectory (establishing a character in a scene, focusing on a target object).

All trajectories indexed by their context are then gathered in a database from which a camera motion graph will be built.

4.3 Real data extraction

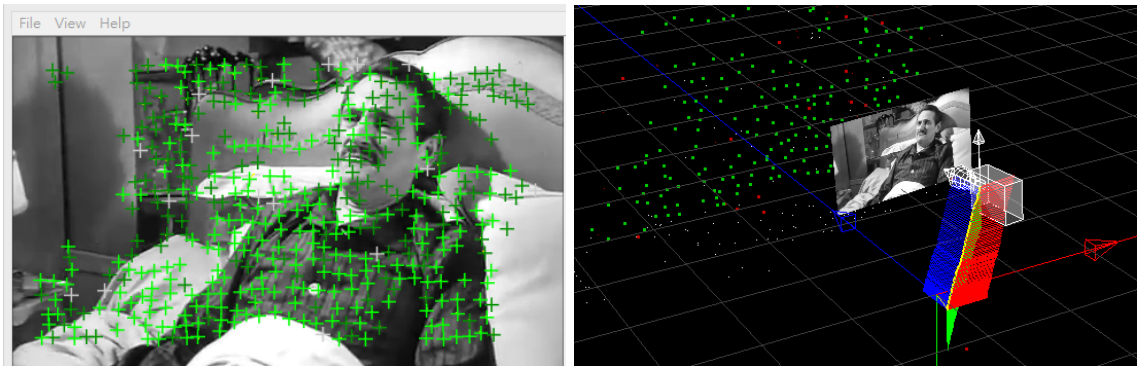


Figure 16: Harris corner detector for feature points of a picture (on the left) that are used to reconstruct the camera trajectory from a sequence a shot (on the right).

In this process, we rely on the Voodoo Camera Tracker software (a tool developed by the University of Hannover). 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 16 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 like KLT tracking or SIFT matching. As a result, the tool estimates the camera trajectory as well as the camera parameters (see Figure 16). The user then manually specifies the 3D location of the target (a couple of targets) involved in the scene – an easy task since Voodoo provides a point cloud representation

of the scene. The trajectories are then converted to the appropriate representation (G_f, G_1 or G_2).

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 together with a display to visualize the viewpoint of the cameraman in the virtual scene. In our approach, a plug-in was written for the Motion Builder tool (an Autodesk animation software) to extract the trajectories.

5 Building the Camera Motion Graph

The construction of a camera motion graph consists in sampling every pair of camera trajectories in order to evaluate the best transitions between the trajectories. One of the main challenges of building such transitions is ensure continuity in the resulting trajectory [Tho09]. We provide the details on how the possible transitions are evaluated and generated. Two types of transitions are considered for each couple of camera configurations $\mathbf{q}_i$ and $\mathbf{q}_j$. Continuous transitions (T^C) represent generated pieces of trajectories that link $\mathbf{q}_i$ and $\mathbf{q}_j$ together. Non-continuous transitions (T^N) represent cuts between configurations $\mathbf{q}_i$ and $\mathbf{q}_j$. 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 shown to be useful in case of short-timed occlusions, the camera switching to a further configuration on the same trajectory.

5.1 Sampling Heuristic

Here the sapling heuristic is simple, we take all the camera position extracted as potential node to a transition. This heuristic allows us to have possible transition on each camera position and that is really helpful when it comes to solve the visibility constrain as having a high frequency of transition on the trajectory allows quick change when the visibility is lost. An other point to consider is that this also make dead end really rare.
In the camera motion graph we consider two types of transitions: continuous transition and cuts. There only one type of transition between two node either a continuous are a cut. When it comes to the transition computation we apply the following heuristic: we first test if it is suitable for the node to have a continuous transition, if not we try with a cuts. This heuristic is based on the assumption that cuts are more frequent kind of transition and that we have to favour the continuous transition over the cuts. An example can be found in figure 17

5.2 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 it 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:

$$c_{JC} = \begin{cases} 0 & : \quad \mathbf{x}_{\mathbf{q}_i}\widehat{\mathbf{x}_\mathbf{o}\mathbf{x}_{\mathbf{q}_j}} > \pi/2 \vee |p(\mathbf{q}_i, h_\mathbf{o}) - p(\mathbf{q}_j, h_\mathbf{o})| > t_h \\ 1 & : \quad otherwise \end{cases}$$

where $p(\mathbf{q}_i, h_\mathbf{o})$ 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 0.20% on a unit screen).

The line of action rule states that the camera must not cross an imaginary line passing through two actors involved in an interaction.

The cost c_{LA} of crossing a line of action establishes whether the two cameras $\mathbf{q}_1$ and $\mathbf{q}_2$ are the

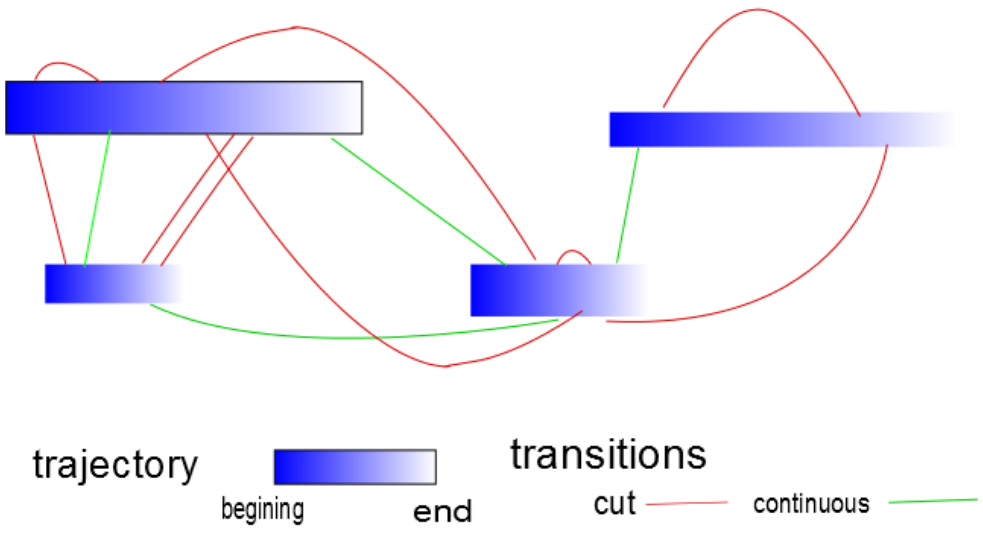


Figure 17: This picture illustrates a camera motion graph. The colour shade represent the time spend by the trajectory. The red line are cuts between two camera poses and green line are continuous transition.

same side of the line $\mathbf{x}_{\mathbf{o}_1}\mathbf{x}_{\mathbf{o}_2}$:

$$c_{LA} = \begin{cases} 0 & : \quad sin(\mathbf{x}_{\mathbf{q}_1}\widehat{\mathbf{x}_{\mathbf{o}_1}\mathbf{x}_{\mathbf{o}_2}}).sin(\mathbf{x}_{\mathbf{q}_2}\widehat{\mathbf{x}_{\mathbf{o}_1}\mathbf{x}_{\mathbf{o}_2}}) > 0 \\ 1 & : \quad otherwise \end{cases}$$

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

5.3 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_θ), and do not follow opposite directions (cost c_{dof}). Furthermore, transitions towards camera configurations too close to the end of their motion should be avoided (cost c_{end}). Costs are expressed as follows:

$$c_d = \begin{cases} 0 & : \quad |\mathbf{x}_{\mathbf{q}_1} - \mathbf{x}_{\mathbf{q}_2}| < 1 \\ |\mathbf{x}_{\mathbf{q}_1} - \mathbf{x}_{\mathbf{q}_2}| & : \quad otherwise \end{cases}$$

$$c_\theta = \begin{cases} 0 & : \quad cos^{-1}(2(\mathbf{u}_{\mathbf{q}_1} \cdot \mathbf{u}_{\mathbf{q}_2})^2 - 1) < \pi/6 \\ 1 & : \quad otherwise \end{cases}$$

$$c_{dof} = \begin{cases} 0 & : \quad |\dot{\mathbf{x}}_{\mathbf{q}_1} - \dot{\mathbf{x}}_{\mathbf{q}_2}| < d_{dof} \end{cases}$$

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

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

6 Using the Motion Graph

Once the camera motion graph is constructed, the run-time process consists in selecting the best nodes in the motion graph and the best transitions with any set of new targets, in a new 3D environment. The process is guided by three strategies: (i) the camera avoids transitions between trajectories when unnecessary (ii) the camera respects the context conveyed by the 3D environment and (iii) the camera tries to avoid the occlusion of targets.

6.1 Context

The camera motion graph encodes a number of possible transitions independently of the context. At run-time, however, possible transitions from a given node need to be re-evaluated to account for changes in the context, such as a change in the actions performed by the target, or a change in the number of targets.

Given that trajectories have been annotated with contexts related to the actions they convey, a simple check is performed. Transitions towards camera paths with unsuitable contexts are not considered as candidates.

In the same way, trajectories encode their number of reference targets. Transitions towards targets with unsuitable number of targets are not considered as candidates.

6.2 Visibility of targets

In order to compute the visibility we use a technique developed by the team.

At run-time, the camera needs to avoid the occlusion of targets. At each frame, their visibility along the current path is evaluated using a ray-casting technique. In order to avoid frequent transitions between trajectories due to partial or short-timed occlusions, the idea of thick paths is introduced, with which to compute the visibility.

A thick path is represented with a volume placed around the immediate future of the camera path (see figure 18, and decomposed into regular cubes. Rather than evaluating the visibility from the camera position to the target, the visibility is evaluated by casting rays from the target towards each cube independently. Occluded cubes are eliminated, the local modifications of the camera trajectory are performed relying on the visible cubes. If no cube is visible, the possible transitions are evaluated to other trajectories in the motion graph.

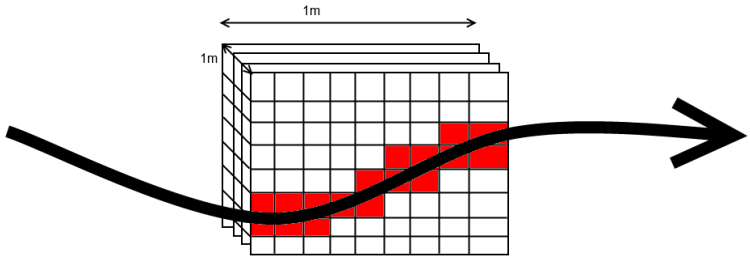


Figure 18: A thick camera trajectory. The arrow represents the camera trajectory, the red cubes are the ones traversed by the camera. All cubes are evaluated for visibility, and the path is locally modified to ensure visibility.

When total occlusion occur, all possible transitions (continuous or cuts) are checked for visibility and the one offering the best visibility is selected. In our visibility algorithm the director can to choose a visibility threshold $\in [0; 1]$. The value influences the frequency of transitions. Furthermore the director is allowed to specify a maximum duration of occlusion so that when the object of interest are hidden by a moving object, the trajectory is not affected and transitions are not selected.

In our framework, the visibility is computed in two steps. Let t (in seconds) be the position in time along one trajectory. The visibility is assessed at two positions in time : $t + t_i$ and $t + t_j$. The overall philosophy of the algorithm is to look far enough ahead $(t + t_j)$ for visibility. If the objects of interest is still visible then the visibility test is not computed between t and t_j (Δt). The position of the objects in the scene is estimated by linear interpolation of the current position. To avoid accumulating too many trajectory errors, the maximum for Δt is 1 second. If the object is not visible in Δt seconds then the visibility is computed in $t + t_i$ seconds.

7 Results and Implementation

7.1 Implementation

Converting the Voodoo text file into a given type of trajectory is made by a parser. This parser takes the Voodoo text file and stores the different parameters of the camera for each frame. Then it is possible to convert the trajectory to a certain type of trajectory, all conversion requires a text file that describes the context's keywords separated by comma. The conversion to a panning camera is straight forward as it only requires to store the orientation of the camera. The conversion to a spherical trajectory requires a VRML file that contain the data of the target followed by the camera. Using those data it is easy to convert the camera parameters into the reference of the target, the size is saved as the scale of the target along the z axis. The conversion to a manifold trajectory requires a VRML file that contains the position of the two targets of the trajectory, then using the result of the article [LC12] it is easy to convert the Cartesian coordinates to the manifold coordinates.

The conversion operation outputs an xml file containing the different information of the trajectory. These xml are then loaded into different classes representing the trajectories. Each class implements an interface called Trajectory that implements the common function and can store the global Cartesian coordinates of the trajectory. Each class has a TrajectoryDescriptor to store the type and the id of the trajectory. The The main method of the different classes of trajectory are the loading function that take an xml file representing a trajectory of the given type and load it into the class containers. The second is a function that converts the trajectory coordinates in the global Cartesian reference frame and store them into the container of the Trajectory class.

The motion graph structure is first computed using the adimensional representation of the trajectories described in Section 4. At this step the motion graph only a list of potential transition between trajectories. The assumption is made for the manifold and spherical trajectories that they have at least one target in common. The transition in this first graph are refined when the trajectories are used in the new scene, at this point we know the final position of the target. The continuous transitions are then computed on-the-fly when they are needed.

Our motion graph was implemented into a the $CryEngine^{TM}$ video game. This engine provides us with a mean to create our scenes with defined actions.

7.2 Result

This section presents some results. Our results are of 2 kinds:

- visibility tests and performance,
- trajectories reproduction.

7.2.1 Visibility tests and performance

To assess visibility we provided our algorithm with a random scene. We sampled paths to have over 50 points that are fully interconnected.

Figure 19 shows some captures of the process. We ran these tests on a Dell with Intel Core I7 CPU 2.60GHz. At no visibility test we had around 60 fps. And while visibility test running we had around 40 fps. Figure 20 shows two examples of invisibility time.

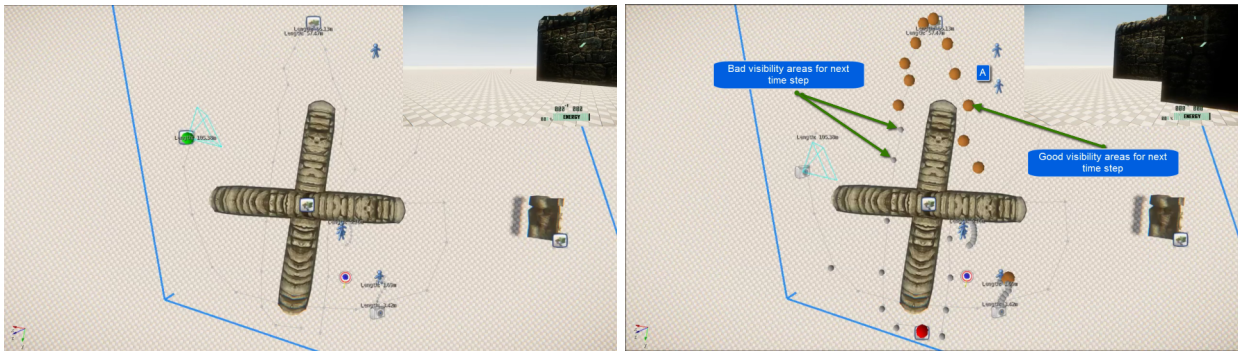


Figure 19: Example scene for visibility computation. The first picture shows the camera green point and the actor A when no visibility is computed. The second shows the same scene with visibility being computed. We can see that all the points are being tested and only a fraction are visible from the actor.

Figure 20: In both these sequences we can see the actor before, during and after occlusion. Here our threshold is set to 0.6 seconds. This means that if the actor is to be invisible for 0.6 seconds or more we change the trajectory. For the two frames we can see that the trajectory does not change for little occlusions since they last respectively 0.3 and 0.4 seconds.

7.2.2 Reproducing trajectories

We present two different results of trajectory reproduction. The first scene is a conversation between two actors. The trajectory used is a manifold trajectory (see Appendix A Figure 21). The first scene is a game scenario we created. The trajectory used is a combination of 2 spherical trajectories (see Appendix **??** Figure 22). We compared these results with the original trajectories.

8 Conclusion

Computer graphics techniques are pushed towards more and more realism. A gap is created between the high quality real-time rendering techniques and the available camera control techniques to convey them. Improving the realism of the camera control, by the use of cinematography, is a promising approach to fill that gap. In this report we have presented different techniques used to solve the problem of camera control. We have distinguished them into three categories: the one that aim at computing good shots, the one which aims at planning camera trajectories and the one which aims at ensuring a good succession of shots. We notice that all those techniques rely on procedural approaches.

Based on those observation we introduce a way to use real camera trajectories in virtual scene. The structure that we proposed is a camera motion graph, inspired by character animation techniques, in order to deal with real data. The trajectories are extracted from video footage, and expressed in a normalized representation and tagged with context elements. The camera motion graph is then computed from these trajectories and applied to a new scene in which we want to control the camera. The camera motion graph consist in node which represent the camera position and the transition are either cuts between shots are continuous transitions between trajectories. At the run-time the camera is controlled following two constraints, visibility is computed on the graph using thick path and ray-casting, to avoid target occlusion, and context tagged are used to discriminate the transition in the graph.

For future work, we propose first to refine the data structure itself, by for instance improving the distance metrics for the transitions. Second, the camera motion graph can be exploited in more advanced ways, for instance to achieve specified communicative goals (ie specified camera position or part of trajectory) defined by the user at a given time of the scene, while maintaining the visibility. In turn, this would require dynamically planning the best path to reach the specified position or part of trajectory, in an evolving context.

References

[AWCO10] Jackie Assa, Lior Wolf, and Daniel Cohen-Or. The virtual director: a correlation-based online viewing of human motion. In *Computer Graphics Forum*, volume 29, pages 595–604. Wiley Online Library, 2010.

[BDGER08] Paolo Burelli, Luca Di Gaspero, Andrea Ermetici, and Roberto Ranon. Virtual camera composition with particle swarm optimization. In *Smart Graphics*, pages 130–141. Springer, 2008.

[BK00] Robert Bohlin and Lydia E Kavraki. A lazy probabilistic roadmap planner for single query path planning. 2000.

[CN05] Marc Christie and Jean-Marie Normand. A semantic space partitioning approach to virtual camera composition. In *Computer Graphics Forum*, volume 24, pages 247–256. Wiley Online Library, 2005.

[CNO12] Marc Christie, J-M Normand, and Patrick Olivier. Occlusion-free camera control for multiple targets. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 59–64. Eurographics Association, 2012.

[CON08] Marc Christie, Patrick Olivier, and Jean-Marie Normand. Camera control in computer graphics. In *Computer Graphics Forum*, volume 27, pages 2197–2218. Wiley Online Library, 2008.

[CT12] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[GSKJ08] Michael Gleicher, Hyun Joon Shin, Lucas Kovar, and Andrew Jepsen. Snap-together motion: assembling run-time animations. In *ACM SIGGRAPH 2008 classes*, page 52. ACM, 2008.

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

[HHS01] Nicolas Halper, Ralf Helbing, and Thomas Strothotte. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. In *Computer Graphics Forum*, volume 20, pages 174–183. Wiley Online Library, 2001.

[HNR68] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[HS88] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. UK, 1988.

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 473–482, New York, NY, USA, 2002. ACM.

[KRE+10] Christian Kurz, Tobias Ritschel, Elmar Eisemann, Thorsten Thormahlen, and H-P Seidel. Camera motion style transfer. In *Visual Media Production (CVMP), 2010 Conference on*, pages 9–16. IEEE, 2010.

[LC08] Tsai-Yen Li and Chung-Chiang Cheng. Real-time camera planning for navigation in virtual environments. In *Smart Graphics*, pages 118–129. Springer, 2008.

[LC12] Christophe Lino and Marc Christie. Efficient composition for virtual camera control. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–70. Eurographics Association, 2012.

[LCR+02] Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. Interactive control of avatars animated with human motion data. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 491–500. ACM, 2002.

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[Mas65] J. Mascelli. *The Five C's of Cinematography: Motion Picture Filming Techniques.* Cine/Grafic Publications, Hollywood, 1965.

[NO03] Dennis Nieuwenhuisen and Mark H. Overmars. Motion planning for camera movements in virtual environments. Technical report, Utrecht University, 2003.

[OSTG09] Thomas Oskam, Robert W Sumner, Nils Thuerey, and Markus Gross. Visibility transition planning for dynamic camera control. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 55–65. ACM, 2009.

[Tho09] R. Thompson. *Grammar of the Edit.* Focal Press, 2009.

[VBP+09] Thales Vieira, Alex Bordignon, Adelailson Peixoto, Geovan Tavares, Hélio Lopes, Luiz Velho, and Thomas Lewiner. Learning good views through intelligent galleries. In *Computer Graphics Forum*, volume 28, pages 717–726. Wiley Online Library, 2009.

[VFSH01] Pere-Pau Vázquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Viewpoint selection using viewpoint entropy. In *VMV*, volume 1, pages 273–280, 2001.

[Zhe13] Changxi Zheng. One-to-many: example-based mesh animation synthesis. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 145–153. ACM, 2013.

Appendix A: Recreated Trajectory

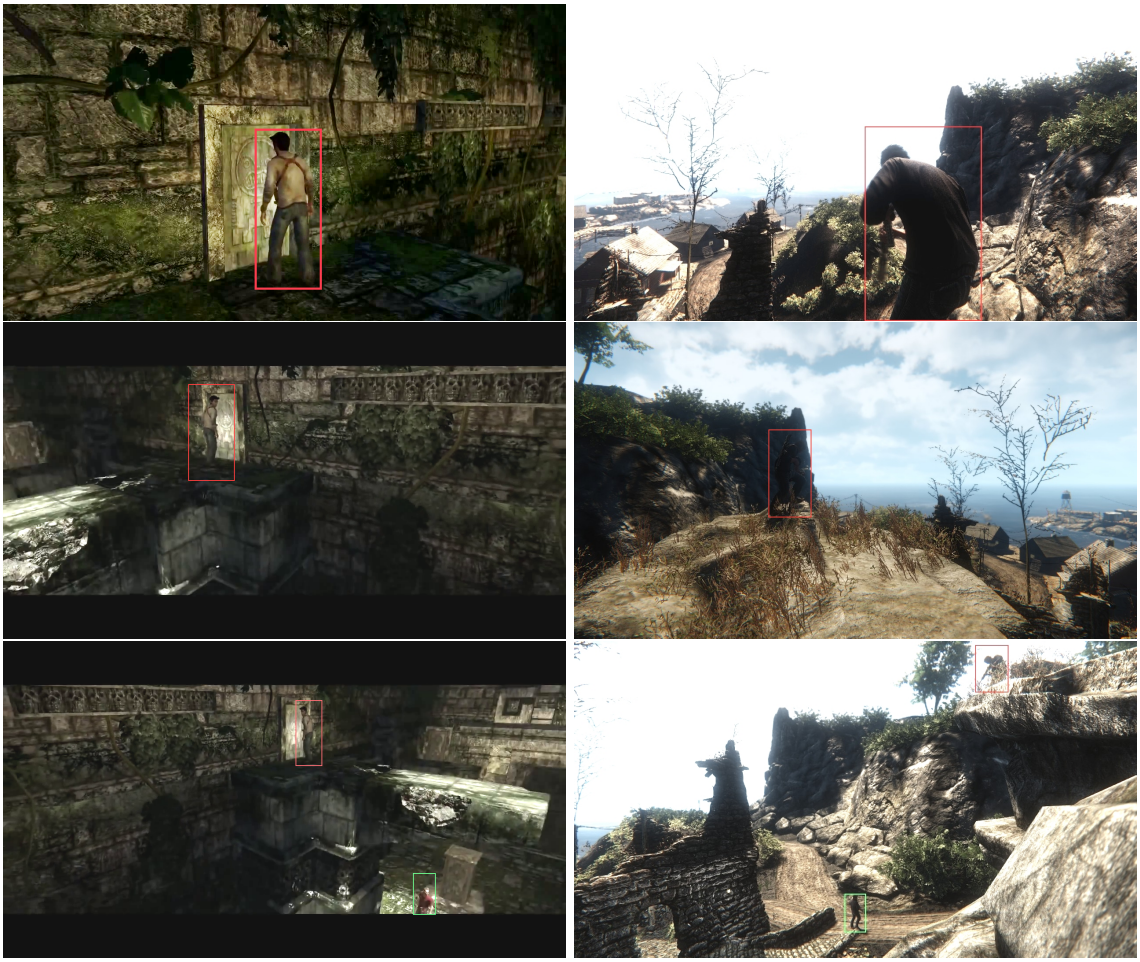


Figure 21: On the left picture we display the captured trajectory and on the right picture we display the computed trajectory. We can see that they are similar even if the actor's configuration is not exactly the same.

Figure 22: In this example the trajectory relies on a manifold representation. Trajectory, and size of actors on the shots are the same despite different a different configuration of the actors.

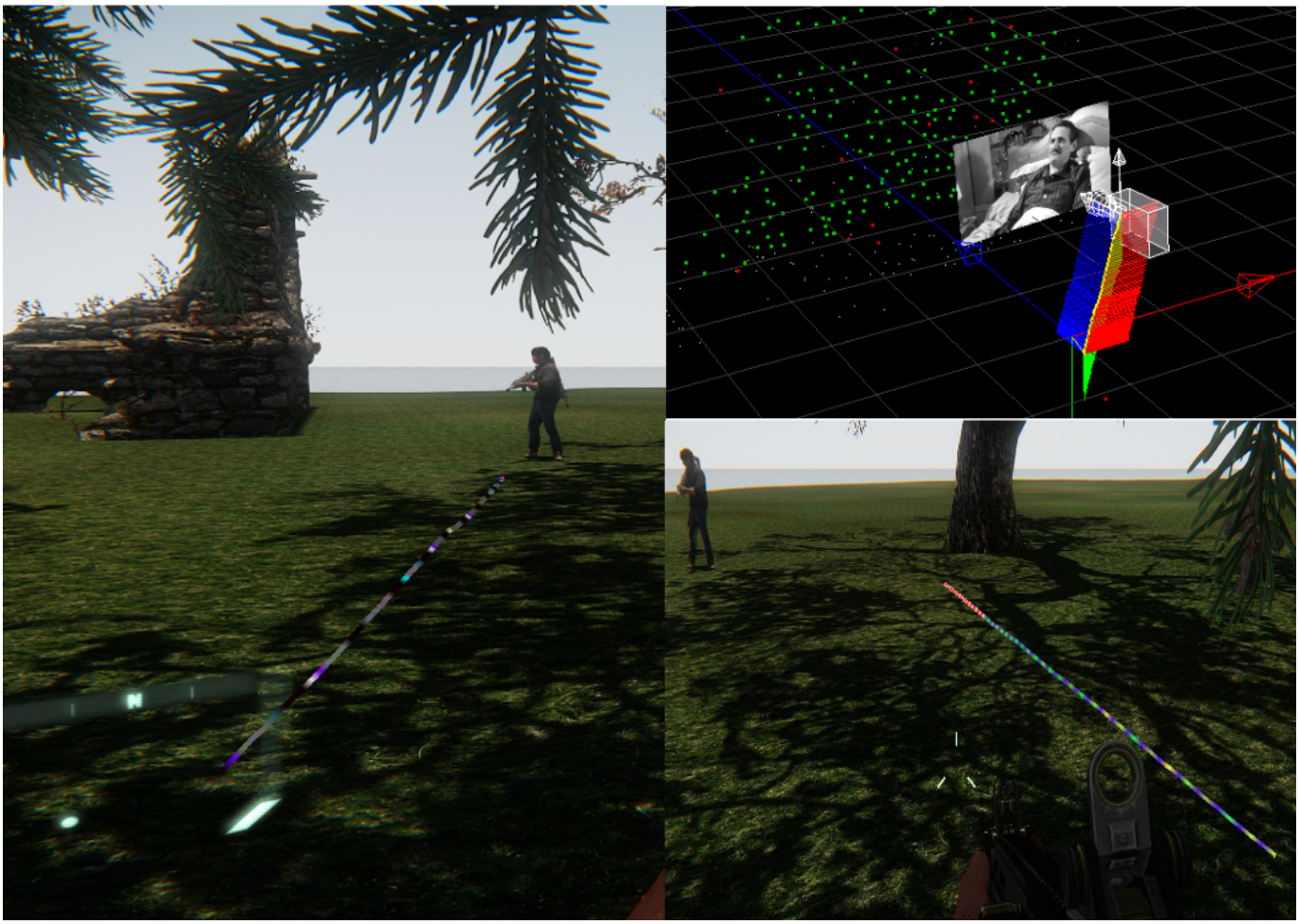


Figure 23: In this example the trajectory relies on a spherical representation. The size of the original target (a head) is smaller than the second (all body) the trajectory is then resize