Automatic camera control meets emergency simulations: An application to aviation safety

Published
DOI:10.1016/j.cag.2015.03.005

Please cite as:
Ranon R., Chittaro L., Buttussi F. Automatic Camera Control meets Emergency Simulations: an Application to Aviation Safety, Computers and Graphics, Vol. 48, May 2015, pp. 23–34.

Automatic Camera Control meets Emergency Simulations: an Application to Aviation Safety

Abstract

Computer-based simulations of emergencies increasingly adopt 3D graphics to visualize results and thus generate complex dynamic 3D scenes with many potentially parallel events that affect large groups of virtual characters. To understand the portrayed scenario, a viewer could interactively control a flying camera or switch among a set of virtual cameras that have been previously placed at modeling time. The first solution imposes a cognitive load on the viewer that can distract him/her from the analysis task, and (s)he might miss events while moving the camera. The second solution requires additional work in the modeling phase, and even a very large number of cameras could fail to correctly frame events because of dynamic occlusions. More sophisticated automatic camera control methods could help, but the methods in the literature are designed for sequential dialogue-like events that involve at most two or three characters and therefore would not work. In this paper, we present a fully automated, real-time system that is able to monitor events in emergency simulations, select relevant events based on user-provided filtering rules, and control a virtual camera such that the events of interest are properly presented to the viewer. To illustrate how the system works in practice, we also describe the first application of automatic camera control to the domain of aviation safety.

Keywords: automatic camera control, emergency simulations, aviation safety

1. Introduction

Computer-based simulations of emergencies are increasingly used for a variety of purposes, including planning, prediction of outcomes, accident investigation, and training. Systems have begun to adopt realistic 3D graphics to visualize simulations results (e.g., [1, 2, 3]), thereby generating complex, dynamic 3D scenes with many potentially parallel events affecting large groups of virtual characters. Presenting the resulting animations to a viewer in an effective manner is thus challenging.

The traditional approach to the visualization of 3D simulations is to place multiple virtual cameras in the scene at modeling time and switch among them at run time to observe the different events that occur. However, as the spatial complexity of the scenario and the number of events increase, even a very large number of cameras could fail to correctly frame many events, e.g., because of dynamic occluders that prevent any of the pre-defined cameras from adequately capturing some of the action. Moreover, manually placing the virtual cameras can require a considerable modeling effort that in general must be repeated for each simulation. Even for the same simulation, multiple camera setups may be necessary based on what features a viewer finds the most interesting. For example, a safety expert could be interested in how the entire emergency egress of a crowd from a building evolves, while a firefighter who uses the same simulation for training would need to focus on details that are relevant to first response duties in the field such as the location and evolution of fires.

An alternative solution is to let the viewer interactively control a flying camera during the simulation. However, this approach imposes a cognitive load on the viewer that can distract him/her from the analysis task and that has the additional disadvantage that (s)he might miss events while moving the camera.

Automatic camera control methods could provide solutions to such problems, thus relieving the user from the burden of manual camera placement, selection, and control. However, most methods that have been proposed in the literature are designed for sequential dialogue-like events involving at most two or three characters and are thus not suited to situations that include several parallel events involving many characters. Indeed, none of these solutions have been adopted for emergency simulations.

In this paper, we present a novel and fully automated, real-time camera control system for emergency simulations that is able to monitor interesting events and present them to a viewer. We propose to organize such a system into two conceptual modules: a *Camera Operator* and a *Director*. The *Camera Operator* is based on extending a recent virtual camera computation approach [4] to calculate, whenever needed, a virtual camera that aims at visualizing the maximum number of currently occurring events of interest. The *Director* then analyzes the virtual cameras that are computed by the *Camera Operator* and chooses which camera to use and when to use it to visualize simulation events to the viewer. To illustrate how the system operates in practice, this paper applies it to a complex case in the domain of aviation safety. However, the system is not limited to aviation and could be utilized in other emergency domains.

The paper is organized as follows. In Section 2, we briefly review past work on computer-based emergency simulations and automatic camera control and motivate the need for the proposed approach. In Section 3, we describe the proposed camera control system, and in Section 4, we apply the system to a full aircraft evacuation scenario that reproduces the main aspects of a well-known recent accident. Finally, in Section 5, we conclude the paper and outline future research directions.

2. Related Work & Motivations

2.1. Computer-based Emergency Simulations

Computer-based emergency simulations are increasingly used for a variety of purposes, including planning, prediction of outcomes, accident investigations, and training. In particular, emergency evacuations have received considerable attention in the literature. Gwynne et al [5] reviewed 22 evacuation models and classified them into three main categories: optimization, simulation and risk assessment. EXODUS is a well-known evacuation model that was successfully applied to analyze both building [6] and mass-transport [7] evacuations. The system includes specialized modules to model very specific aspects such as (i) the characteristics of occupants (e.g., age, gender, and physical disabilities), (ii) their movements and behaviors, and (iii) the physiological impact of toxicity due to smoke. A variant of EXODUS, called airEXODUS [8], is specifically tailored to aircraft evacuation.

In general, EXODUS and the other evacuation models attempt to precisely compute the values of several variables to predict an evacuation outcome or analyze a real case, but they do not focus on real-time interaction (e.g., interactions that dynamically affect an evolving simulation) and have limited visualization features (e.g., 2D maps or simplified 3D models). In particular, the EXODUS system can be used with vrEXODUS, which is a 3D visualizer of the simulations that operates as a graphics post-processor of previously generated simulations. The Glasgow Evacuation Simulator [9] introduced the possibility of opening or closing routes in real time to test different evacuation paths. Moreover, the simulator supports the visualization of an evacuation using CAD-CAM 3D models of buildings, but occupants are represented only by colored cylinders.

In addition to advancing the simulation domain, improving the realism of graphics and real-time interaction with the simulator would extend the application of evacuation models to training, thus allowing trainees to learn by directly interacting with virtual objects and characters. Moreover, with the help of 3D animations, trainees could virtually experience emergency scenarios that are difficult, expensive and dangerous to reproduce in the real world, thereby getting a better understanding of complex scenarios and cause-effect relationships [10, 11]. Systems that employ realistic 3D graphics consider various emergency scenarios such as car accidents with fire and toxic gas propagation in road tunnels [1], smoke hazards in subway stations and schools [12], fire drills in buildings [13], and evacuations of airports [3] and nuclear facilities [2].

2.2. Automatic Camera Control

Current approaches to the visualization of the 3D simulations discussed in the previous section are based on either placing virtual cameras in the scene at modeling time, and switching between them at run time, or manually controlling a moving camera at run time. However, depending on the spatial complexity of the scenario, even a very large number of cameras or a very skilled manual control will fail to correctly frame certain events, e.g., because of dynamic occluders. Moreover, manually placing the cameras can involve a considerable modeling effort, which in general has to be repeated for each simulation, and manual camera control in real time imposes a cognitive load on the viewer that can distract him/her from the analysis task. Many emergency simulations involve hundreds (or even thousands, as in simulations of the 9/11 attack [14]) of independent characters and many different types of events that are occurring in parallel over an area that could be very large. As a result, it is very hard to select and visualize all of the relevant details of such emergencies with the camera control approaches of current simulators.

Automatic camera control methods could offer a method of addressing these issues. In the following, we analyze the main aspects that an automatic camera control system must consider to present a simulation. For each aspect, we briefly discuss the state of the art and illustrate why current approaches are not adequate for emergency simulations.

The first fundamental aspect is how to find virtual cameras that ensure the visibility of events of interest. Current automatic control approaches can be organized into two main categories: approaches that search for virtual cameras anywhere in the scene and that can consider an arbitrary number of targets [4, 15, 16], hereinafter called *global solvers*, and approaches that focus on ensuring the continuous visibility of one [17, 18] or a few [19, 20] dynamic targets and that search only in a region around a current camera. In both approaches, visibility is typically defined in terms of a combination of various visual properties such as target screen size, occlusion, and angle from which the target is observed. In emergency simulations, events might occur anywhere in the scene; therefore, the ability to find virtual cameras anywhere in the scene is substantially more important than ensuring continuity in visualizing the simulation. Unfortunately, most global solvers typically suffer from performance issues because they rely on stochastic optimization strategies (e.g., population-based algorithms) to sample the search space. An exception is a recent proposal by Ranon and Urli [4], who introduced more effective candidate camera initialization and evaluation strategies whereby a single virtual camera can be computed in tenths of milliseconds (instead of hundreds) in quite complex scenes.

The ability to find cameras that can frame various current events of interest is only the first step toward the broader goal of conveying meaning (or at least making it inferable) to a viewer. This topic has been the subject of several research papers, e.g., [21, 22, 23, 24], that focus on narrative events and mimic the language of films by encoding cinematographic rules such as typical shots and continuity editing. However, such approaches are limited to film dialogue-based interactions among two or three characters and to consider one event at each time. For example, the *Virtual Cinematographer* [23] and the FILM system [24] are able to film events in real time by selecting among a set of *idioms*. An idiom contains information about the number of targets, shot types and, in the *Virtual Cinematographer*, the timing of transitions between shots to best communicate events, such as three virtual actors conversing, as they unfold. Camera placement is selected among a few pre-encoded, idiom-specific alternatives, e.g., depending on the targets' visibility. However, there is no guarantee that, in a spatially complex environment, any of the alternatives will provide a suitable framing of targets. Lino et al [21] improved upon the two above-mentioned systems by considering a narrative event and computing a set of *director volumes*, i.e., volumes in the scene that encode both shot type and visibility information for the considered event. Then, they searched the director volumes for optimal virtual cameras that guarantee continuity when cutting between cameras and selected the best virtual camera based on style elements. In this approach, the visibility computations are performed in 2D (therefore, they are not applicable to multi-level scenes or small objects) and do not consider dynamic occluders such as other characters in the scene. In summary, current systems based on cinematographic rules work well in situations where spatial complexity is limited, the scene is mostly static, and camera control targets consists of mainly two or three characters that are engaged in dialogue-like events. Moreover, such systems can typically consider only one event at a time. Due to these limitations, they are poorly suited to emergency simulations.

A system that better addresses the needs of emergency simulations was proposed by Galvane et al [25], who focused on presenting events that occur in crowd simulations. Their system relies on Reynolds' model of steering behaviors to control and locally coordinate a collection of camera agents in real time in a manner similar to a group of reporters. Camera agents can be either in a scouting mode, thereby searching for relevant events to present, or in a tracking mode, thereby following one or more unfolding events. The system was tested using a crowd simulation with 100 virtual characters in an exterior environment, where it provided a good coverage of events (mainly measured as the ratio between observed versus missed events). Compared to our method, their camera control approach has various advantages and disadvantages. The main advantage of their method is that it directly provides smooth camera motions in contrast to the static cameras that we use, which is a feature that might be preferable when the result should exhibit cinematic-like qualities. Moreover, the approach of using a population of cameras naturally enables the simultaneous coverage of events that occur at different locations in the simulation or the coverage of the same event from different perspectives. The disadvantage of moving cameras through smooth motion (no "teleport") is that it might take some time before a camera is able to reach the position where an event occurs. Because this duration depends on where the cameras are at the moment of event occurrence, camera movement time is not predictable. Increasing the number of cameras might help, but this would also increase the computational complexity (their paper reports 15 fps for 30 cameras). Moreover, their approach provides a better performance and was demonstrated using exterior scenes, where cameras can observe and easily detect events that occur without occlusions. For interior or mixed interior/exterior scenes that are typical of emergency evacuations, their camera computation methods would likely take more time than in purely exterior scenes to discover events because they would not be able to detect an event unless it is visible. Finally, their approach does not consider how to select the camera to present events to the viewer.

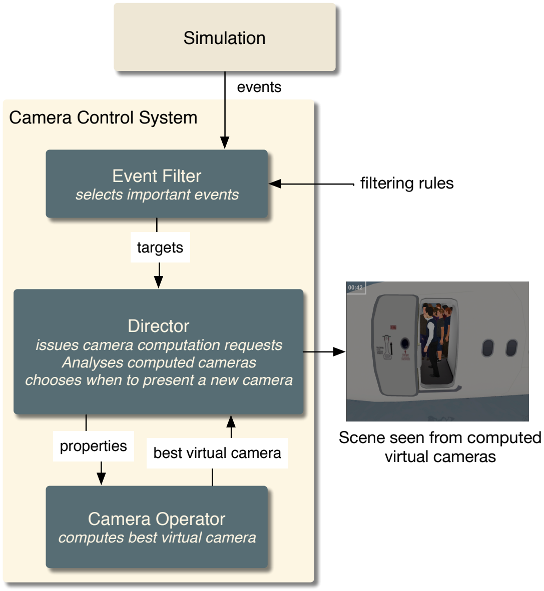


Figure 1: Overview of our system.

3. The proposed system

In this section, we present a system that can monitor events from a simulation, select interesting ones based on user preferences, and present the events to a viewer. Events refer to objects in the 3D scene, which are considered as targets that should be visualized. Our system can operate in real time (i.e., while the simulation updates the 3D scene) without any assumption or pre-processing of the spatial environment or the behavior and shape of 3D objects. The system does not attempt to present events using a cinematic language (e.g., preserving continuity between cuts); the system uses only static virtual cameras because they are sufficient for monitoring a simulation.

To illustrate how the system works in practice, we will apply it to the domain of aviation safety. However, the system is not limited to aviation and can be reused in other emergency domains.

An overview of our system is provided in Figure 1. The simulation sends a stream of all events that occur to the camera control system. The events are then filtered in real time (on the basis of user-provided *filtering rules*) to select the ones that are relevant to the current viewer. For each event that passes the filtering phase, the system extracts *targets*, i.e., objects in the simulation that are involved in the event. Every few seconds, the *Director* module takes the list of *targets* that have been extracted so far and asks the *Camera Operator* module to compute a virtual camera that visualizes the targets by creating a list of properties that the desired virtual camera should satisfy. The *Camera Operator* then tries to determine the virtual camera that best satisfies the request and returns the result to the *Director*, which evaluates the result and decides if and when to present it to the viewer by activating a transition from the current camera to the new camera. In the following, we describe in detail the major activities performed by our system.

3.1. The Event Filter Module

For each event that occurs, the running simulation sends an event description to the camera control system. An event description is a (*subject, action, object*) triplet where

- *subject* is the acting object in the simulation event (e.g., "*Flight Assistant 1*");
- *action* is a textual description of the action performed by the subject (e.g., "*starts opening door*"); and
- *object* is the object in the simulation that is affected by the action (e.g., "*door 1L*");

As explained in Section 1, viewers are typically interested in a subset of events, and the subset varies according to the purposes for which a simulation is run. To select the subset of events, the viewer provides a list of strings, each of which can be the name of an object in the simulation or (part of) an action. *Filtering rules* then perform substring matching between the list of strings and the stream of events output from the simulator. For example, the list of strings ("*Flight Assistant 1*", "*door*") will match all events where *Flight Assistant 1* or any *door* are involved. Matching events are then parsed, and simulation objects contained in them are inserted in the *targets* list, which is accessed by the *Director* module.

Because a simulation might, at times, not generate events that match, the camera control system allows the viewer to specify a set of targets that should be framed in the absence of interesting events. For example, one could specify the entire scene if (s)he wants the camera control system to search for global overview shots when no events match his/her interests.

3.2. The Camera Operator Module

The *Camera Operator* module is based on a recent open-source declarative virtual camera computation library developed by Ranon and Urli [4], which is able to compute in a given amount of time the virtual camera that best satisfies a list of visual properties. The visual properties can express desired values of the size (area, width or height), visibility, camera angle and on-screen position for any choice of objects in the 3D scene. From an input list of visual properties, the library first builds a function that returns a numeric value indicating to what extent a given virtual camera satisfies the properties. Then, a solver based on Particle Swarm Optimization [26] iteratively searches the 3D scene for the virtual camera that maximizes the satisfaction function. The library works with any type of scene or object and does not require any preprocessing of the scene; the library relies on the 3D rendering engine to obtain information about the bounding volumes of objects and to perform ray casting queries to measure visibility. A solution can be returned in any amount of time, although in complex scenes, additional computation time will generally translate into a better solution (i.e., greater satisfaction of visual properties). We refer the reader to [4] for a detailed explanation of the optimization approach[1].

In this paper, we need to define a set of properties that characterize any virtual camera that can frame a specific list of targets, thereby making them prominent in the images rendered from the camera and ultimately allowing a viewer to understand the events that involve them. Our proposal is to use the following properties for each target:

- the screen area of the *target* should preferably be at least 10% of the screen size when we only have one target; in this way, it will be the main or one of the main subjects in the displayed image so that the viewer can easily recognize it, and the viewer will also see objects around it, to understand its position in the scene. When there are more targets, the value is divided by their number;
- the *target* should be fully visible (no other objects occluding it);
- the *target* should be framed as close as possible to the center of the screen (so that the viewer's attention is drawn to it);
- the *target* should be viewed from the front. The front is defined based on the category of the object. For example, in the case of a character, this means being able to see the character's face.
- the *target* should be viewed from a medium to high angle (we want to avoid viewing the target from too high or too low because in such cases, it could be difficult to understand what the target is doing).

Note that certain properties are more important than others. For example, the visibility of a target is clearly more important than framing it in the center of the screen because it is certainly preferable to be able to see a target, even close to the screen edge, than to not be able to see it because of occlusions caused by other objects.

In the adopted virtual camera computation library, the above requests are implemented by a `Size`, `Occlusion`, `Framing`, and two `Angle` properties with the target as first argument. The second argument of each property is a satisfaction function that returns values in [0,1] (where 0 means no satisfaction and 1 means full satisfaction) expressed as a linear spline. Table 1 presents the visual properties that we have associated to each target and their corresponding satisfaction functions. For example, for the `Size` property, we have defined the spline in terms of the points (0,0), (0.05, 0.01), (0.08, 0.8), (0.1,1) and (1,1),

[1] Source code of the library is available at http://bit.ly/1wdBOqq

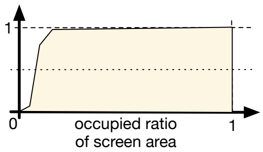
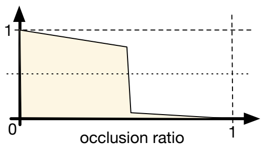
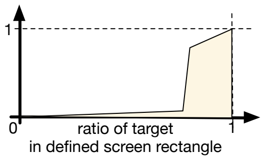
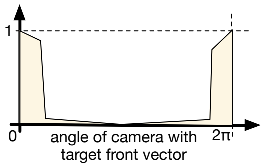
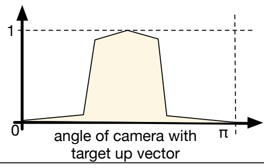
Property Type	Semantics	Weight	Satisfaction Function
`Size`	the target object should cover at least 10% of the screen area, or less if there are multiple target objects	2.5	
`Occlusion`	the target should not be occluded by other objects	4.0	
`Framing`	the target object should be framed inside a screen rectangle with minimum and maximum corners equal to (0.2,0.2) and (0.8, 0.8), respectively, in viewport coordinates	1.0	
`Angle`	camera in front of the object	1.0	
`Angle`	camera parallel or slightly above object	1.5	

Table 1: Properties defined for each target object to compute a virtual camera. Weights reflect relative importance of properties and have been determined empirically. Slight variations of weights would not alter substantially the result.

where the x value in the function is the ratio of the screen area that is occupied by the rendered target and the y value is the corresponding satisfaction value. The weight of each property (shown in the table) is a number that reflects the relative importance of the property compared to the other properties. For example, the weight of the `Occlusion` property is four-times larger than the weight of the `Framing` property. Because the satisfaction of a virtual camera is defined as a weighted sum of all the property satisfaction functions, this means that in cases when both framing and visibility cannot be guaranteed, the optimization process will prefer visibility. Weights have been empirically determined by running a few simulations, and slight variations do not alter substantially the result.

In certain situations, it might be preferable to use other sets of properties. For example, the viewer might be more interested in visualizing an overview of passengers exiting from the doors on a plane. In this case, the goal is not to obtain virtual cameras that are sufficiently close to recognize a character but to derive virtual cameras that visualize groups of characters in different positions in the scene. This can be expressed by requiring, for each target, a minimum screen area that is much lower than 10% and to soften requirements concerning angle (so that, for example, top-down virtual cameras also satisfy the requirements). In our system, viewers can choose (and even modify their choice while the simulation is running) whether to promote *target recognizability* (i.e., the *Camera Operator* will try to frame targets at a close distance using all of the above properties) or *event coverage* (i.e., the *Camera Operator* will try to frame targets at a greater distance if this is necessary to capture more targets using the more relaxed properties described above). In the last case, the properties are modified as follows: for each target, the minimum screen area becomes 0.5%, the `Angle` property that considers the target front vector is removed (so that it is equally satisfying to frame the target from behind) and the `Angle` property that considers the target up vector considers as satisfying any angle from 0 to 45 degrees.

As with all virtual camera computation approaches based on optimization, there is no guarantee that the best returned virtual camera will satisfy all of the given properties. First, such a camera might not exist; e.g., consider a case in which we are simultaneously interested in two targets that are located in two opposite zones of an aircraft: the pilot's cockpit and the rear galley. These situations are not unlikely because, in principle, the simulation might compute events that occur simultaneously in completely different locations. In general, our system will try to find the virtual camera that frames more targets because this corresponds to greater satisfaction. In the example of the two targets in opposite zones of an aircraft, the system will have to choose one of the targets and miss the other (which could be framed later). A more subtle issue is the case where only certain properties of a target can be satisfied. For example, it might be impossible to find a camera that guarantees both the required size and (at least partial) visibility. In this case, while the returned camera satisfies certain target properties, it may not allow the viewer to understand the events that involve the target. More generally, due to the limited time available for virtual camera computation and the stochastic nature of search, the *Camera Operator* might at times be unable to find a virtual camera that satisfies all properties or even all of the properties for some targets even if such a virtual camera exists. In general, as the geometrical complexity of the scene and/or the number of targets increase, this type of issues are more likely to occur. An increase in geometrical complexity typically translates into more time required to explore the scene in search of a camera that satisfies the visibility properties. A larger number of properties increases the time required to evaluate the satisfaction of virtual cameras during the search process. We address all these issues by evaluating the virtual camera that is computed by the *Camera Operator* before using it to visualize events to the viewer.

3.3. The Director Module

The *Director* module manages the entire camera control process. This module decides which camera is shown to the viewer, how and when to transition to a new camera, issues virtual camera computation requests to the *Camera Operator* and evaluates the returned virtual camera. The *Director* module is executed at regular time intervals (0.2 seconds), and its operation is schematized in Figure 2.

When it is time to change the camera to be shown to the viewer, the *Director* takes the current *targets* list from the *Event Filter* module, computes the list of properties, issues a virtual camera computation request to the *Camera Operator*, and stops its current execution. If instead this operation was performed during the previous execution, the *Director* would take the virtual camera that meanwhile has been computed by the *Camera Operator*, evaluates it and decides if the camera should be used. In such a scenario,the *targets* list is emptied.

The evaluation of the virtual camera returned by the *Camera Operator* considers which of the targets are effectively framed, i.e., the involved events are recognizable. We define a target as effectively framed by a virtual camera if its `Size` and `Occlusion` properties have a minimum satisfaction value of 0.5 and 0.3, respectively, out of 1. This corresponds to the target being half of the preferred minimum screen area and half visible. For the other properties, we rely on the virtual camera computation process to maximize the satisfaction, but we also accept virtual cameras that do not frame certain targets in the screen center or with the required angle because these requirements could likely be difficult or impossible to satisfy for multiple targets simultaneously. Therefore, for each virtual camera, we compute two lists of targets: targets that are effectively framed (*framed targets*) and frames that are not effectively framed (*missed targets*). The evaluation has a negligible computational cost because it was previously performed during the search process. A virtual camera is deemed to be good for the viewer when its *framed target* list contains at least one target. If a virtual camera is not good, it is discarded. When the *Camera Operator* fails to find a good camera in the allowed time frame, the *Director* immediately issues a new virtual camera computation request with one target removed from the *target* list (preferably one of the targets that are already framed by the current camera or a random target). The result of the virtual camera computation request will be available for evaluation in the next execution of the *Director* module. Targets in the virtual camera *missed target* list, if present, will be considered as targets for the next virtual camera computation if no interesting events are detected in the next camera update cycle.

A good virtual camera, before being used, is compared to the current camera. It is not always necessary to transition to a new camera; there are times in which new interesting events involve targets that the current virtual camera is already framing, or a newly computed virtual camera does not provide significantly more information than does the current one. For this reason, before changing the current virtual camera, we compare it with the new candidate camera. If the new virtual camera frames more or different targets compared to the current camera, we transition to it. If they frame the same targets, then we transition to the new camera only if its satisfaction value is greater than the current value by at least 5%. If the newly found camera frames only a subset of the targets that the current camera is framing, we maintain the current camera.

The frequency of transitions to a virtual camera that visualizes new events is a critical choice. To maximize the coverage of events, we should compute a new virtual camera and possibly transition to it as soon as the event passes the filtering stage. However, in simulations of emergencies wherein many events can occur in a short amount of time, this could result in multiple virtual camera changes per second, which would make it impossible for the viewer to understand what is occurring.

General rules of cinematography dictate that static shots should last between 2 and 10 seconds. In our system, the viewer can set the minimum time between virtual camera transitions. In testing our system on aircraft accident scenarios, we have empirically noted that a time of 3-4 seconds is a good compromise between event coverage and comprehension.

Transitions are currently implemented as straight cuts. While this solution has the disadvantage that it takes a bit of time for viewers to understand the camera changes, allocating time for the camera to transition from the old location to the new location could make viewers miss events. However, when the new camera is close to the current camera in terms of position and orientation, a smooth transition may be better for the viewer.

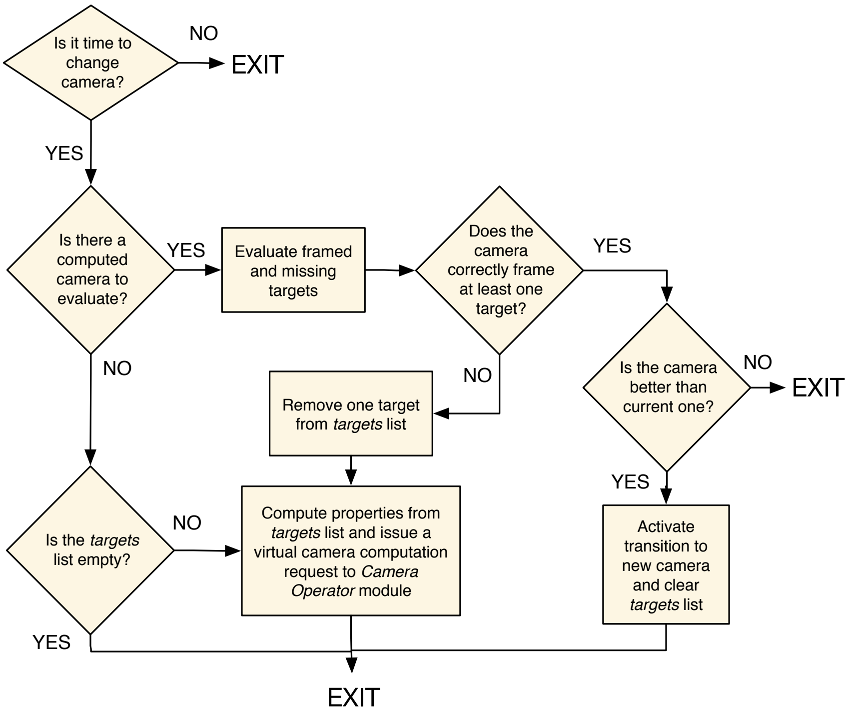


Figure 2: Functioning of the *Director* module.

4. Results

We have extensively tested our system using simulations of different types of aircraft emergencies. In particular, the examples presented in this section concern a full aircraft water landing (ditching, in aviation terminology) and evacuation scenario. More precisely, we employ an accurate 3D reconstruction of an Airbus 320 [27], one of the most common aircraft types in service. The reproduced accident is very similar to the accident involving US Airways flight 1549 [28]: a few minutes after take-off, the aircraft suddenly loses thrust in both engines due to a severe strike with a flock of large birds and is forced to ditch because the lack of thrust makes it impossible to reach nearby airports.

The 146 virtual passengers in the simulation can perform several autonomous tasks, which include the following: (i) fastening seat belts as soon as the airborne plane shows signs of instability, (ii) maintaining the brace position during ditching until the plane comes to a stop, (iii) reaching for the nearest exit, (iv) locating an alternative exit in the presence of exits that cannot be used (e.g., in the following examples, the rear exits are not usable because they are below the water level), (v) opening overwing doors, (vi) exiting the aircraft using a level exit or an overwing exit, and (vii) going toward the bottom of a slide raft and sitting on it. Moreover, the simulation includes three virtual flight assistants that can perform three additional tasks: (i) open floor level doors, (ii) order passengers to stand back until a raft is fully inflated, and (iii) block unusable exits and redirect passengers. In the examples presented in this section, two flight assistants help passengers at the front exits, while the other attendant blocks the two unusable rear doors and redirects passengers to the front and overwing exits until all passengers are away from the flooded rear galley.

Each passenger and flight assistant has a unique name (e.g., ”Passenger 113” or ”Flight Assistant 1”) in the simulation, and all aircraft-relevant parts are labeled (e.g., ”door 1L” or ”door 2L”). These names and labels are used as subjects and/or objects in the event triplets. The event actions concern all of the tasks described above as well as changes in states of the aircraft doors (e.g., closing and opening) and slides (e.g., inflating).

The total number of events in each of the following examples is 1829. The first 590 events (e.g., fastening seat belts and assuming a brace position) occur in the 4 minutes and 20 seconds during which the aircraft is airborne. The other 1239 events occur during the evacuation, which lasts approximately 2 minutes and 30 seconds. In the following examples, we will focus on the evacuation because this phase contains a large number of events in a limited amount of time; thus, it presents a greater challenge to the camera control system. Note that, due to the stochastic nature of particle swarm search, the system can generate different cameras in different runs even if the simulation and its events are identical.

We describe three examples of system use. For each example, we describe the scenario and provide sample screenshots. In addition, to enable the reader to see first-hand the actual output of the system, we have included a video as additional paper materials[2]. Both the examples presented in this Section and the video use a frequency of camera transition of 4 seconds. In Section 4.1, we analyse the performance of the system in these scenarios.

The first example (at minutes from 00:18 to 02:03 in the accompanying video) considers the perspective of flight assistant training, in which trainers and trainees are highly interested in observing the behavior of the crew. Therefore, we specify "flight assistant" as a matching string in the filtering rules. The *Event Filter* module will then discard all events that do not match this string while selecting all evacuation events concerning flight assistants (30 in our example). As a result, the *Camera Operator* module will receive requests to frame one, two, or three flight assistants simultaneously (depending on the timing of events) as well as the object that they could interacting with (e.g., doors). Because it is important to frame the flight assistants at close distances in this example, we set the system to prefer target recognizability over event coverage. Figure 3 shows six of the 30 cameras that were computed and used by our system using these settings during the entire simulation.

More precisely, immediately after the impact, all three flight attendants simultaneously stand up. Because two of them are at the front exits and one is at the rear exits, it is impossible to simultaneously frame all of them, and the system finds a camera showing the two at the front, as shown in Figure 3a. One of the flight assistants at the front exits is the first to reach and open a door, as shown by the camera in Figure 3b. When the second flight assistant at the front exits reaches and opens the other front door, both flight assistants order passengers to stand back until the slides are fully inflated. In this case, our system finds a camera that frames both subjects (Figure 3c). When a front slide is fully inflated, the system shows the nearby flight attendant stepping aside and indicating the way to passengers, as shown in Figure 3d. In contrast, the flight assistant at the rear exits is sending passengers away because water is entering the rear galley (Figure 3e). Only when all passengers have left the rear galley can the flight attendant move forward (Figure 3f) and exit the aircraft (Figure 3g). When all passengers have exited the aircraft, the other flight assistants can exit (Figure 3h).

The second and third examples consider the perspective of an aircraft designer or an accident investigator, who are interested in observing how, where and when passengers and crew exit an aircraft in an accident. In this case, we specify "exit" as a matching string in the filtering rules. The *Event Filter* module then selects all exit events (149 in our case, one for each of the 146 passengers plus three for the flight assistants). Exit events begin at the exact instant a passenger exits the plane and last about two seconds. As a result, the *Camera Operator* module will often receive requests to simultaneously frame many passengers (depending on the timing of exit events) and mainly focus on doors. In this case, different viewers can be interested in observing the exit behaviour with different priorities: some viewers may be interested in watching passengers and exits at a close distance, while other viewers may be more interested in understanding the egress as a whole. Therefore, in our second example (at minutes from 02:05 to 03:29 in the accompanying video), we set the system to prefer target recognizability, while in the third example (at minutes from 03:30 to the end in the accompanying video), we set the system to prefer event coverage.

If the system is set to prefer target recognizability, when the first passengers begin to exit on the left wing, the camera correctly focuses on them (Figure 4a). A few seconds later, passengers start to use the right overwing exits. In this case, there is no camera that can simultaneously frame all of the overwing exits and the exiting passengers while preserving recognizability; therefore, the Director module can choose to continue showing passengers exiting on the left wing or switch to the right wing. Figure 4b shows the second choice. When the front right raft is fully inflated and passengers start using the front right exit, our system can find cameras that frame both front and overwing right exits (Figure 4c). When all front and overwing exits are available, at each camera update, the system computes the position and angle that maximize the properties shown in Table 1 for the highest number of targets (Figure 4d and 4e). In particular, when only one exit is used, the system can compute a new camera to frame only it (Figure 4f).

If event coverage is preferred instead, the system will find more distant cameras that can frame more targets. More precisely, at the beginning of the simulation, when there are passengers only exiting from the left overwing exits, the camera will specifically focus on them, as in Figure 5a, but when passengers begin to use the right overwing exits, the system will try to find a camera that can simultaneously frame passengers at all of the overwing exits, as in Figure 5b. When the right raft is inflated and passengers start to use it, the system computes a camera that focuses on exits at the right side but continues to frame passengers exiting from the left overwing exits (Figure 5c). Finally, when the left raft is also available, the cameras will try to simultaneously frame all used exits, as in Figure 5d. As in the previous example, when only a subset of exits is used at a particular moment in the simulation, the system will compute a more focused camera, as in Figure 5e, but the different settings will also include cameras that simultaneously cover events that are occurring at the two opposite sides of the aircraft (Figure 5f), despite partially reducing recognizability.

While an extensive and formal evaluation with domain experts has not yet been performed, we have informally tested the system by using the videos it creates as a means of communication with aviation professionals (researchers and pilots) as well as individuals who are unfamiliar with aviation (students and researchers in other domains). In each case, we first informed the viewer about the general goals of the videos (i.e., the kind of events that the camera control system was instructed to frame). Then, we showed the video without any comment or verbal explanation. Finally, we discussed about the aircraft accident and evacuation depicted by the video to check if there were any comprehension issues or doubts concerning the important events. From this purely informal experience, the output produced by the system appears to be effective: the videos were clearly understood without ambiguities and the events are effectively presented. A possible issue that emerged is that some-

[2] The video is also available at http://youtu.be/DJq87oasil8

Figure 3: Various cameras from the ditching simulation when the filtering rules specify to match "flight attendant" events and the preferences are set to target recognizability.

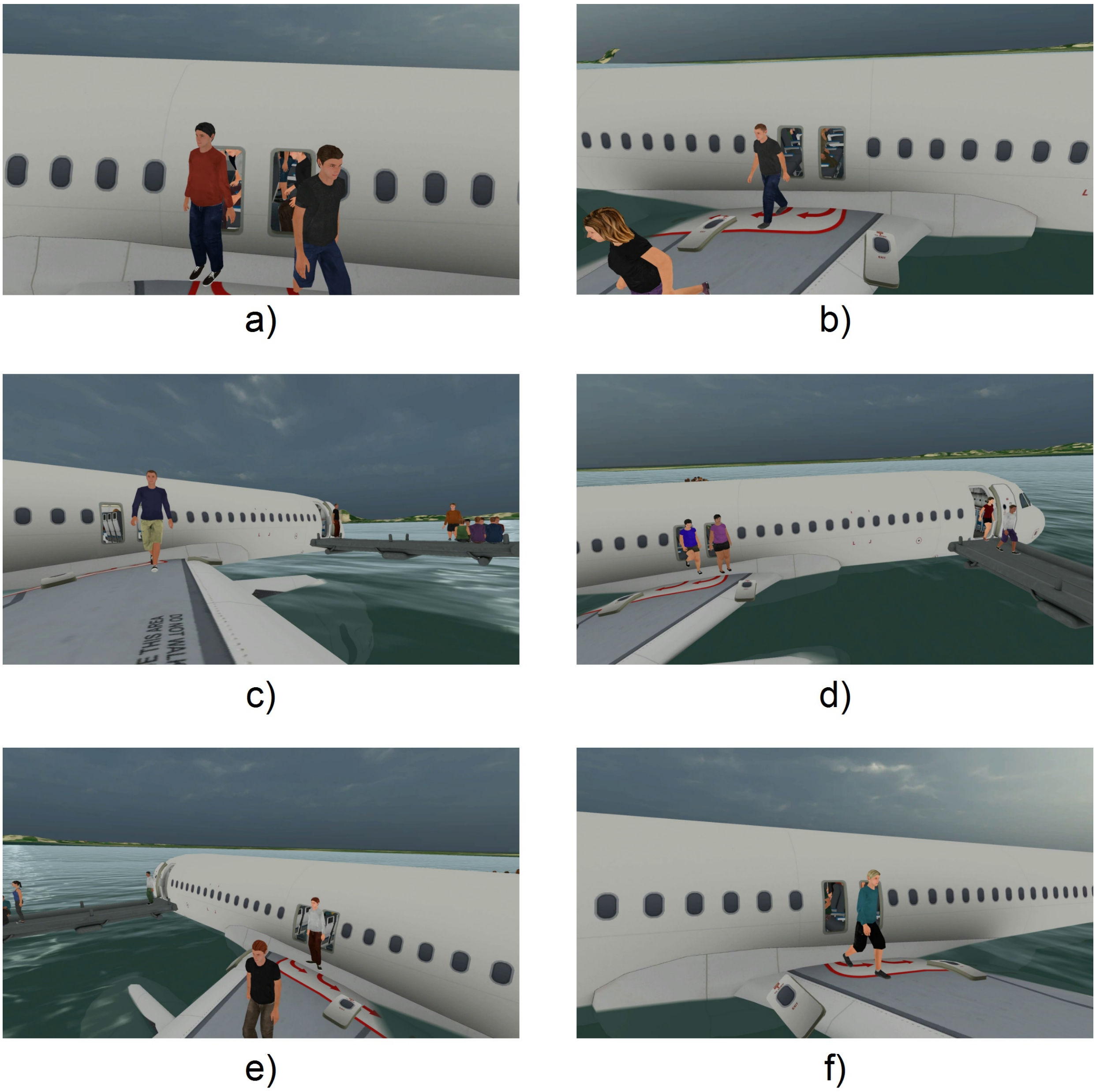


Figure 4: Various camera shots from the ditching simulation when the filtering rules specify to match “exit” events and the settings prefer target recognizability over event coverage.

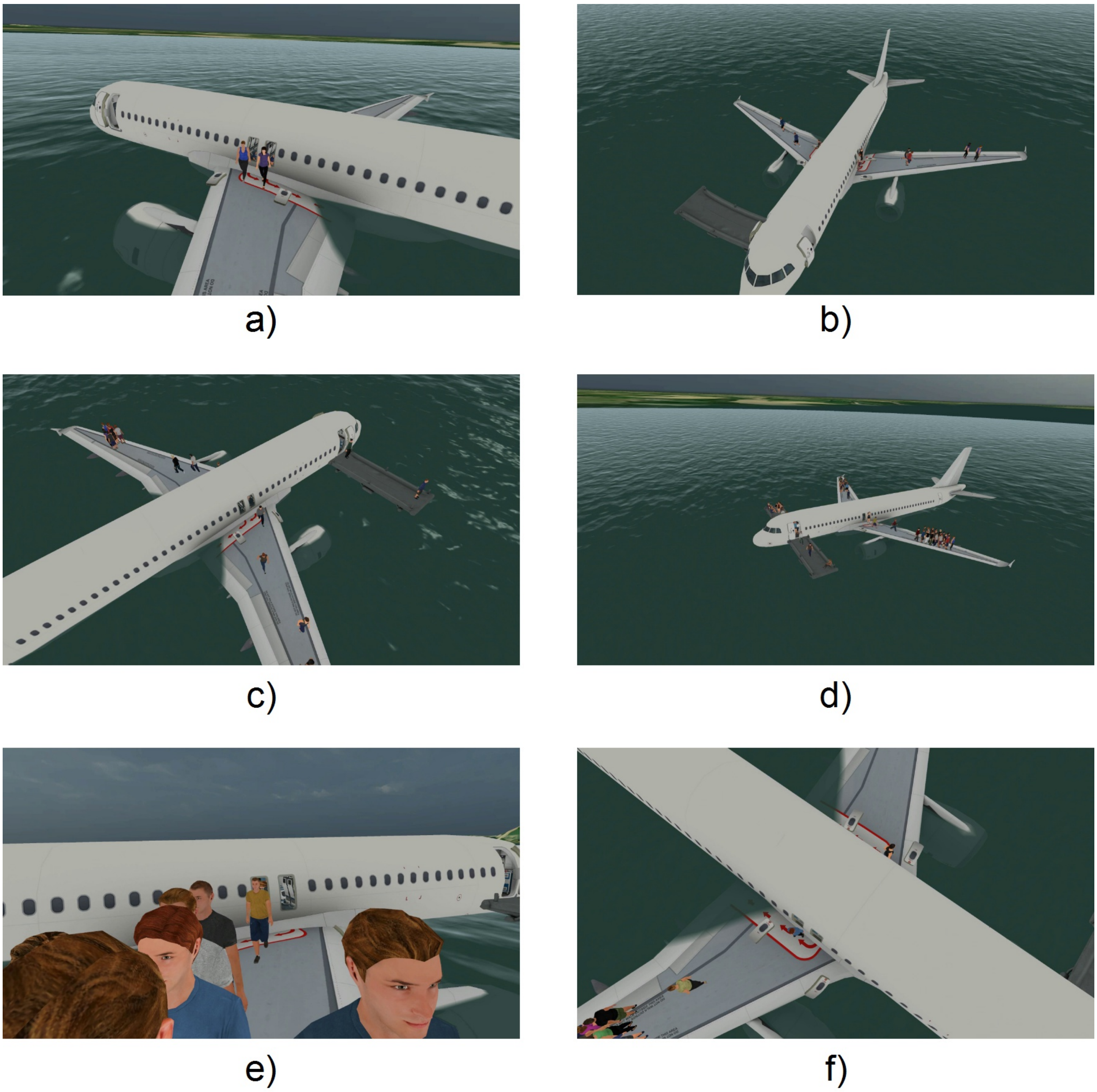


Figure 5: Various camera shots from the ditching simulation when the filtering rules specify to match “exit” events and the settings prefer event coverage over target recognizability.

times the change in camera position and orientation required a few instants for the user to reorient herself, although this issue concerns mainly viewers who are not familiar with the detailed internal and external structure of an aircraft. The system proved also very useful while working on the simulation visualization, because it provided a way for developers to focus on specific parts of the simulation and check for graphical glitches.

4.1. Implementation and Performance

The scenarios presented above are implemented in C# using the Unity 4 game engine [29]. The camera control system, as well as the simulation, are implemented in C# as a Unity scripts, and due to Unity limitations, run on the same thread on the CPU (i.e., they cannot run in parallel). Since computing and rendering the simulation is already demanding on the CPU, the computational cost of the camera control system should be as low as possible. By far, its most expensive activity is the computation of cameras in the *Camera Operator* module, whose allotted time, as explained in Section 3.2, is a parameter that can be set. However, in deciding its value, one faces two contradicting objectives. On the one hand, by allowing more time to the *Camera Operator*, we increase the probability of finding more satisfying solutions (i.e., cameras that better frame events or frame more of them); on the other hand, more time means, in CPU-bound applications like our simulation, decreasing the frame rate. For example, in our simulation, if we set the time available to the *Camera Operator* to 30 milliseconds, this means that each time a camera needs to be computed, the frame rate will drop drastically as the CPU cost for each simulation frame (without the camera control) is already, on average, around 25 milliseconds.

To help with this issue, we can split the computation of a camera among a few consecutive frames to limit its impact on frame rate, with the only drawback of delaying the presentation of the result by a few frames. Figure 6 illustrates the performance, in the scenarios described in the previous Section, of three different choices about the time for computing a new camera: 7 milliseconds in one frame, 14 milliseconds equally split among two frames, and 21 milliseconds equally split among three frames. To measure the performance, we use the number of framed events, as the average frame rate is similar in all cases. By looking at the box plots, it is clear that there is a general increase in the number of framed events, for all quartiles, both by going from 7 milliseconds to 14, and from 14 to 21. The increase is more notable in the exit scenarios, which are more complex in terms of average number of targets to frame. Note also that, in the exit scenario, by preferring event coverage over target recognizability, we greatly increase the median number of framed events (by around 65% in the condition with 21 milliseconds).

We tried also a fourth condition with 28 milliseconds split in four frames, but it did not result in any significant increase in performance. As explained in the previous Section, the reason is that in all scenarios events happen very often in parallel and in different parts of the plane. Therefore, no camera, regardless of how much time we spend in computing it, can frame them together. In the flight assistants scenario, for example, this happens because one of the flight assistants stays in the back of the plane for most of the simulation, while the other stay located in the front part of the plane. In the other scenarios, passengers exit at the same time from doors that are located at both sides of the plane, and, especially when target recognizability is set, only one door can be framed at each time. When event coverage is selected, instead, the system manages to find cameras that frame passengers at longer distances, exiting from different doors (e.g. all doors on the same side of the plane).

From this and other experimental activity we performed, we can draw some indications on how to set the time available to the *Camera Operator* module. First, one should choose a time that is compatible with a target frame rate. Then, one can multiply the chosen time by a number of frames, so that the total computation time will guarantee good results in the scenario at hand. More specifically, the total time is mostly a function of the maximum number of targets that needs to be framed at the same time (in our examples, a time budget of 21 milliseconds was enough for 10-12 targets). Finally, the number of consecutive frames over which a camera computation is carried out should be limited since delaying too much the presentation of the newly found camera might cause some brief events to be missed.

All data were obtained on a 2.3 GHz Intel Core i7 processor with 16 GB RAM and an NVidia GeForce FT 750 M. With this machine, the simulation, including the camera control system, runs at between 30 and 60 fps, depending on the number of animated characters displayed, with an average of 40 fps. As explained above, the frame rate is largely dictated by the simulation, as the cost of the camera control system is at most 7 milliseconds per frame for a few consecutive frames for the *Camera Operator* module, plus the operations of the *Director* module, which cost about 1 millisecond and is executed every 0.2 seconds. Figure 7 shows the milliseconds spent by simulation, rendering and camera control code executed on the CPU over a period of 300 frames in one of the exit scenarios presented above. Green bars refer to rendering preparation calls, which take the majority of time because of the large number of animated characters. The bright orange and bright cyan bars refer to the *Camera Operator* module, which is executed every few seconds, with a maximum cost of around 7 milliseconds per frame (the two bars respectively refer to the cost of the called method, and the cost of the subcalls). The zoomed image in the top right part of the Figure shows instead, in red, the cost of the *Director* module, which is practically negligible compared to rendering preparation.

5. Conclusions and Future Work

In this paper, we have presented a novel application of automatic camera control to emergency simulations and demonstrated the system on a detailed aviation case study.

Our system extends a recent virtual camera computation approach to extract interesting events from a simulation, solve virtual camera computation problems, analyze the results, and determine the virtual camera used to present events of interest to a viewer. As shown in Section 4, our method allows us to visualize different aspects of aircraft evacuation scenarios without

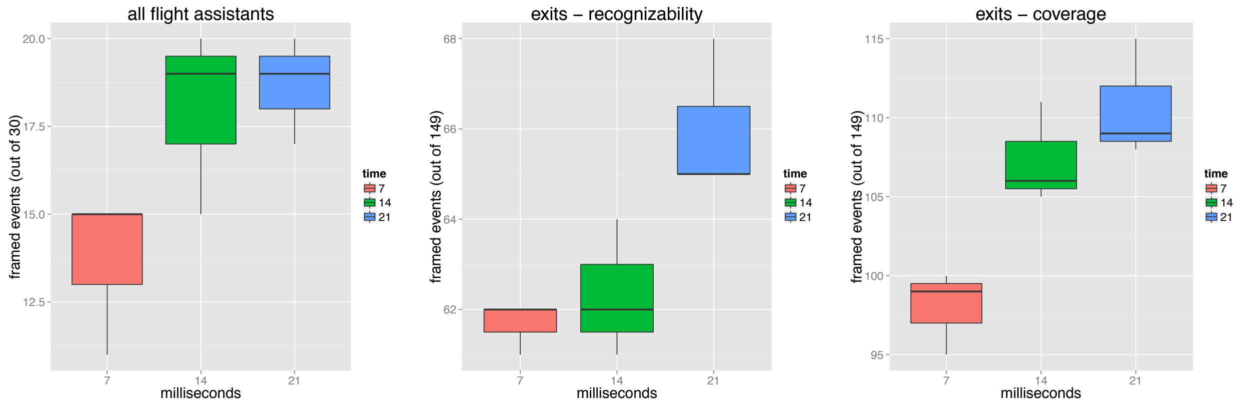


Figure 6: Distribution of the number of framed events in the considered scenarios, with three different time budgets for computing cameras: 7 ms in one frame, 14 ms in two consecutive frames (7 per frame), and 21 ms in three consecutive frames (7 per frame). Data obtained with 50 runs per condition.

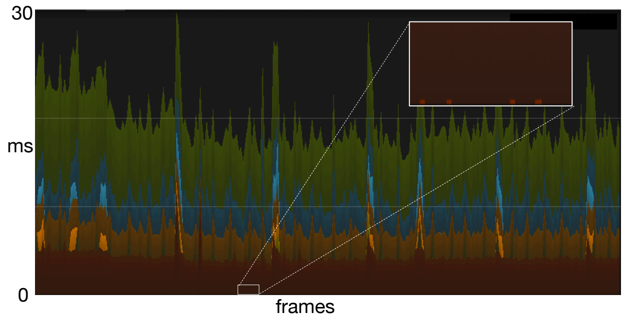


Figure 7: Milliseconds spent by simulation, rendering and camera control code in one of our scenarios over a period of 300 frames. Green bars refer to rendering calls; bright orange and cyan bars refer to the *Camera Operator* module. Bright red bars (zoomed in in the top-right of the image) refer to the *Director* module.

any camera modeling, programming, or control effort by the user. The system is not aviation specific, and could be applied to other safety domains. In particular, one of our next research directions is to apply the system to fire emergencies in buildings. Another interesting potential field of application is video games testing, particularly multi-player games, and perhaps in addition to automatically collected metrics [30]. To this end, it would be interesting to improve event filtering such that more sophisticated rules can be expressed, for example, based on the temporal relations between events.

We plan to conduct a formal evaluation of the system with aviation professionals. However, carrying out a formal experiment in which the system is contrasted to a control condition (simulator without automatic camera control) would probably create an unfair comparison. It would indeed require the user to take charge of camera control in the non-automated condition and, from our own experience, the workload that manual camera control generates makes it difficult to follow the events with the same ease as automatic camera control. Moreover, when several events happen very closely in time, it can be even impossible for the user to manually follow them.

We also plan to improve the camera control method. A straightforward extension would be the possibility of simultaneously computing and visualizing more cameras when one camera is not sufficient to cover current events. This could be implemented by simply requesting the *Camera Operator* module to immediately compute additional virtual cameras when the current virtual camera is missing certain targets and by setting such targets as the ones to be framed. A more general solution would be to change the virtual camera computation algorithm such that the algorithm is able to return multiple solutions instead of only one, i.e., considering the virtual camera computation problem as multi-objective optimization. However, to the best of our knowledge, no camera control approaches with such capabilities have been developed.

A final interesting issue is the addition of high-level knowledge in the virtual camera computation and selection process. Currently, the system reacts to events that are occurring at the moment of changing the current camera without attempting to establish a correlation between past and present events based on their meaning. An ideal visualization of an emergency simulation should instead be able to derive causal relationships between events and perform virtual cameras computation and editing such that these relationships are effectively conveyed to the viewer.

References

[1] Cha M, Han S, Lee J, Choi B. A virtual reality based fire training simulator integrated with fire dynamics data. Fire Safety Journal 2012;50:12–24.

[2] Mól ACA, Jorge CAF, Couto PM. Using a Game Engine for VR Simulations in Evacuation Planning. IEEE Computer Graphics and Applications 2008;28(3):6–12.

[3] Tsai J, Fridman N, Bowring E, Brown M, Epstein S, Kaminka G, et al. ESCAPES: evacuation simulation with children, authorities, parents, emotions, and social comparison. In: The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. International Foundation for Autonomous Agents and Multiagent Systems; 2011, p. 457–64.

[4] Ranon R, Urli T. Improving the efficiency of viewpoint composition. IEEE Transactions on Visualization and Computer Graphics 2014;20(5):795–807.

[5] Gwynne S, Galea ER, Owen M, Lawrence PJ, Filippidis L. A review of the methodologies used in the computer simulation of evacuation from the built environment. Building and Environment 1999;34(6):741–9.

[6] Owen M, Galea ER, Lawrence PJ. The Exodus Evacuation Model Applied To Building Evacuation Scenarios. Journal of Fire Protection Engineering 1996;8(2):65–84.

[7] Galea E, Perez Galparsoro J. A computer-based simulation model for the prediction of evacuation from mass-transport vehicles. Fire Safety Journal 1994;22(4):341–66.

[8] Galea ER, Blake SJ, Lawrence PJ. The airEXODUS evacuation model and its application to aircraft safety. In: International Aircraft Fire and Cabin Safety Research Conference. 2001,.

[9] Johnson CW. Using evacuation simulations for contingency planning to enhance the security and safety of the 2012 olympic venues. Safety Science 2008;46(2):302–22.

[10] Chittaro L, Ranon R. Web3D technologies in learning, education and training: Motivations, issues, opportunities. Computers & Education 2007;49(1):3–18.

[11] Guttormsen Schär S, Krueger H. Using new learning technologies with multimedia. IEEE MultiMedia 2000;7(3):40–51.

[12] Xu Z, Lu X, Guan H, Chen C, a.Z. Ren . A virtual reality based fire training simulator with smoke hazard assessment capacity. Advances in Engineering Software 2014;68:1–8.

[13] Smith SP, Trenholme D. Rapid prototyping a virtual fire drill environment using computer game technology. Fire Safety Journal 2009;44(4):559–69.

[14] Johnson CW. Applying the lessons of the attack on the world trade center, 11th September 2001, to the design and use of interactive evacuation simulations. In: Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05. New York, New York, USA: ACM Press; 2005, p. 651–60.

[15] Olivier P, Halper N, Pickering JH, Luna P. Visual Composition as Optimisation. In: Artificial Intelligence and Simulation of Behaviour. 1999, p. 22–30.

[16] Bares WH, McDermott S, Boudreaux C, Thainimit S. Virtual 3D camera composition from frame constraints. In: Proceedings of the eighth ACM international conference on Multimedia. ACM Press; 2000, p. 177–86.

[17] Li TY, Cheng CC. Real-time camera planning for navigation in virtual environments. In: Butz A, Fisher B, Krger A, Olivier P, Christie M, editors. Smart Graphics; vol. 5166 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-540-85410-4; 2008, p. 118–29.

[18] Oskam T, Sumner RW, Thuerey N, Gross M. Visibility transition planning for dynamic camera control. In: 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '09; vol. 1. New York, New York, USA: ACM Press; 2009, p. 55–65.

[19] Christie M, Normand J, Olivier P. Occlusion-free Camera Control for Multiple Targets. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Eurographics Association; 2012, p. 59–64.

[20] Halper N, Helbing R, Strothotte T. A Camera Engine for Computer Games: Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence. Computer Graphics Forum 2001;20(3):174–83.

[21] Lino C, Christie M, Lamarche F, Schofield G, Olivier P. A Real-time Cinematography System for Interactive 3D Environments. In: 2010 ACM SIGGRAPH / Eurographics Symposium on Computer Animation. 2010, p. 139–48.

[22] Hornung A, Lakemeyer G, Trogemann G. An Autonomous Real-Time Camera Agent for Interactive Narratives and Games. In: Proceedings of the IVA 2003: 4th International Working Conference on Virtual Agents. Irsee, Germany: Springer-Verlag; 2003, p. 236–.

[23] He Lw, Cohen MF, Salesin DH. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In: SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM Press; 1996, p. 217–24.

[24] Amerson D, Kime S, Young RM. Real-time cinematic camera control for interactive narratives. In: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology. Valencia, Spain: ACM Press; 2005, p. 369–.

[25] Galvane Q, Christie M, Ronfard R, Lim CK, Cani MP. Steering Behaviors for Autonomous Cameras. In: Proceedings of the Motion on Games - MIG '13. New York, New York, USA: ACM Press; 2013, p. 71–80.

[26] Eberhart RC, Kennedy J. Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks 1995; vol. 4. 1995, p. 1942–8.

[27] Airbus . A320 Airplane Characteristics for Airport Planning, Technical Manual. 2014. Http://www.airbus.com/support/maintenance-engineering/technical-data/aircraft-characteristics/ (last access on 25 July 2014).

[28] National Transportation Safety Board . Loss of Thrust in Both Engines After Encountering a Flock of Birds and Subsequent Ditching on the Hudson River, US Airways Flight 1549, Airbus A320-214, N106US, Weehawken, New Jersey, January 15, 2009. Aircraft Accident Report NTSB/AAR-10 /03. Tech. Rep.; National Transportation Safety Board; Washington, DC, USA; 2010.

[29] Unity Technologies . Unity 4. 2014. Http://unity3d.com/ (last accessed on 25 July 2014).

[30] Moura D, el Nasr MS, Shaw CD. Visualizing and understanding players' behavior in video games: Discovering patterns and supporting aggregation and comparison. In: ACM SIGGRAPH 2011 Game Papers. SIGGRAPH '11; New York, NY, USA: ACM. ISBN 978-1-4503-0970-7; 2011, p. 1–6.