Automatized Summarization of Multiplayer Games

Peter Mindek*
Vienna University of Technology
Institute of Computer Graphics and Algorithms

Ladislav Čmolík†
Czech Technical University in Prague
Faculty of Electrical Engineering

Ivan Viola‡
Vienna University of Technology
Institute of Computer Graphics and Algorithms

Eduard Gröller§
Vienna University of Technology
Institute of Computer Graphics and Algorithms

Stefan Bruckner¶
University of Bergen
Department of Informatics

Abstract

We present a novel method for creating automatized gameplay dramatization of multiplayer video games. The dramatization serves as a visual form of guidance through dynamic 3D scenes with multiple foci, typical for such games. Our goal is to convey interesting aspects of the gameplay by animated sequences creating a summary of events which occurred during the game. Our technique is based on processing many cameras, which we refer to as a flock of cameras, and events captured during the gameplay, which we organize into a so-called event graph. Each camera has a lifespan with a certain time interval and its parameters such as position or look-up vector are changing over time. Additionally, during its lifespan each camera is assigned an importance function, which is dependent on the significance of the structures that are being captured by the camera. The images captured by the cameras are composed into a single continuous video using a set of operators based on cinematographic effects. The sequence of operators is selected by traversing the event graph and looking for specific patterns corresponding to the respective operators. In this way, a large number of cameras can be processed to generate an informative visual story presenting the gameplay. Our compositing approach supports insets of camera views to account for several important cameras simultaneously. Additionally, we create seamless transitions between individual selected camera views in order to preserve temporal continuity, which helps the user to follow the virtual story of the gameplay.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: animation, storytelling, game visualization

1 Introduction

Multiplayer video games often contain complex scenes with multiple dynamic foci. The scenes and the game events are viewed by the individual players from different positions. The view of each player is relevant to their part of the gameplay. If an observer watches the players' views, or some additional overview provided by cameras observing the scene, it is not trivial to see the big picture of what has happened during the game. It is likely that some important events will be missed, because they occurred while the observer watched a different view at that time, since the foci are possibly overlapping in time but not in space. However, it is possible to arrange the individual camera views into a story summarizing the gameplay and conveying the big picture.

There are various situations where composing a summary of the gameplay might be beneficial. State-of-the-art game consoles include functionality for the recording of the gameplay and sharing it on social media. There are online services, such as twitch.tv [Twitch Interactive, Inc. 2011], where users can stream gameplays of various video games. There are also tournaments in multiplayer video games, where the summaries of individual matches are created to be shown to a wider audience. The summaries are also used by the participating teams to analyze the gameplay in order to design strategies, detect mistakes made by individual players, and improve their overall performance. There is also a practice of using video games for creating cinematic productions by navigating the game characters according to a pre-defined screenplay, called *Machinima* [Berkeley 2006]. These developments demonstrate that there is a demand for tools aiding in the creation of gameplay summaries or visual narratives.

To create a visual story of a gameplay, various cinematographic effects can be used. However, some specifics regarding multiplayer games have to be taken into account in order to produce a meaningful story. The very fast pace of multiplayer games, as well as many concurrent events require that hard cuts are avoided whenever possible. Instead, it is essential to provide linkage cues in the portrayal of the individual events, such as continuous camera transitions when portraying a sequence of events occurring at different locations.

The aim of this work is to design a method for creating animated summaries of gameplays captured by many dynamic cameras, as in multiplayer video games. The summaries have to portray many dynamic events, possibly occurring at the same time, to facilitate easy understanding of the game action. During the gameplay, the cameras and important game events are recorded. Afterwards, a summary is created which shows the views from the cameras in such a way that the displayed events are meaningfully linked together and they compose a coherent story narrating the gameplay.

By integrating events associated with individual players, and by smoothly transitioning between them rather than splitting the gameplay into separate segments, our method provides an output suitable

*e-mail: mindek@cg.tuwien.ac.at
†e-mail: cmolikl@fel.cvut.cz
‡e-mail: viola@cg.tuwien.ac.at
§e-mail: groeller@cg.tuwien.ac.at
¶e-mail: stefan.bruckner@uib.no

for analyzing the gameplay itself. Aspects such as interaction between players or whole teams, and causal dependencies of individual events are hard to convey by conventional methods, such as sequentially showing the views of all players.

To efficiently portray the linkage of the events, we record their causality, as defined by the game rules and implied by the gameplay. Using this information, we arrange the events into a structure called *event graph*. Additionally, each event is assigned a camera which captures it. This is possible since the events are linked to the players and/or their locations, which are in turn linked to the individual cameras. The event graph together with the linkage between the events and the corresponding cameras contain sufficient information to reconstruct the story which visually narrates the gameplay.

In addition to describing the gameplay by the event graph, we propose two concepts: *flock of cameras* and *ManyCams*. The flock of cameras is a set of cameras whose parameters change dynamically over time. Each camera has a specified lifespan. During its lifespan, a continuous importance function is assigned to the camera. It describes the importance of the corresponding captured events in time. The flock of cameras is a structure which holds all cameras relevant to the gameplay.

ManyCams is a method for selecting the most interesting views captured by the flock of cameras. By merging these views, and inserting smooth transitions between them, a story is created which visually narrates the gameplay. The generated video is a linearization of the gameplay, which has a parallel structure with multiple events occurring at different spatial locations, possibly overlapping in time. ManyCams uses the event graph and the flock of cameras, which are constructed during the gameplay, to create a video summary for a post-mortem visual analysis of the course of the game.

2 Related Work

There are various techniques for integrating visual data into a condensed form which can be presented as a story. Correa and Ma [2010] present a method for creating a compact dynamic narrative from videos. These narratives remove redundancy by using the common background for different stages of motion of various objects in the video. They allow the viewer to quickly inspect the content of the video. Barnes et al. [2010] present a method for creating continuous images summarizing video clips. The method dynamically changes the level of temporal detail by continuous zooming. The generated image shows key features, which would otherwise have to be inspected by watching the entire video. In contrast to these techniques, which summarize linear videos into more condensed forms, our proposed technique summarizes complex parallel structures of multiplayer gameplays by linearizing them into a single summary video.

Various approaches for creating visual stories and presenting data have been developed. Segel and Heer [2010] review different types of narrative visualizations for presenting data based on approaches used in news media. Gershon and Page [2001] describe how storytelling helps to convey information so that it is more easily understood and remembered by the target audience, while Ma et al. [2012] provide an overview of how storytelling can be incorporated into scientific visualization techniques. Wohlfart and Hauser [2007] propose a method for storytelling in volume visualization. In their method, storytelling and story authoring are two distinct steps. In the story authoring step, the dataset is explored by an expert user. While the data are being explored, the expert user iteratively creates the virtual story. In the storytelling step, the findings from the previous step are presented. The user watching the story also has the possibility to interrupt it and perform data exploration on his own. Yu et al. [2010] propose a system which extracts events from time-varying datasets and organizes them into an event graph. By traversing the event graph, an animation with a narrative structure is created which shows the events in a meaningful order. We also employ the concept of event graphs to organize the events extracted from the video game. However, we use it to merge views captured by the flock of cameras into a continuous animated sequence. This allows us to illustrate dynamic scenes of the video games with multiple temporally-overlapping foci. Viola et al. [2006] propose a technique for focusing on selected features of a volume dataset. A viewpoint showing the selected feature in the most informative way is determined. An animation switches between viewpoints for different features. This technique is complementary to our method, since we only focus on compiling the views into a coherent summary, not on selecting adequate viewpoints for individual cameras to capture the scene or the dataset in a certain way.

Creating summaries of computer games has been examined as well. Halper and Masuch [2003] propose a method for extracting game events by analyzing computer game variables to evaluate how interesting individual time-steps of the gameplay are. The authors present several methods for merging these events into scenes, which then compose the story of the gameplay. Cheong at al. [2008] present *ViGLS* - a system for visualizing gameplay summaries from game logs. The game logs are analyzed and sequences of summary actions are extracted from them. These are then sent to a game engine, which replays the actions to generate a visual summary. The goal of our work is to extend these approaches by providing visual links between individual events so that multiple concurrent foci occurring in the dynamic game scene can be captured by the composed summary. Our method linearizes the gameplay, so that continuous dramatizations with a parallel structure can be created. Therefore, our method is able to create summaries of complex gameplays, such as those of multiplayer games, where events are observed by multiple cameras, and where it is important to illustrate causality between the events.

An important aspect of creating visual narratives and informative overviews of 3D scenes is the virtual camera setup. An extensive overview of camera control in computer graphics is given by Christie et al. [2008]. Löffelmann and Gröller [1996] propose extended cameras. By using extended cameras, it is possible to render images with non-realistic perspectives. This enables various effects, such as the seamless combination of several different views of a virtual object in the same image. Agrawala et al. [2000] use multiple projections for different objects in the scene to create various artistic effects. Hsu et al. [2011] present a framework for using several arbitrary cameras to render a single scene at different levels of detail. The cameras are combined to create a single multiscale rendering of the dataset. For this purpose, a mask is specified to define which part of the image should be generated by which camera. Seamless transitions between these image parts are then created.

He et al. [1996] present *Virtual Cinematographer*, a system for the automatic setup of cameras to capture events in dynamic 3D environments according to cinematographic principles. A similar system, utilizing an agent-based camera, is proposed by Hornung et al. [2003]. Oskam et al. [2010] propose a method for planning collision-free camera-paths between two points in a 3D scene, so that a selected focus point is visible during the camera transition. Such camera-shot planning-systems can be also used with our method, since we do not assume any particular camera-placement method for capturing events of the game.

Virtual cameras have been employed for different types of tasks as well. For instance, Qureshi and Terzopoulos [2006] propose

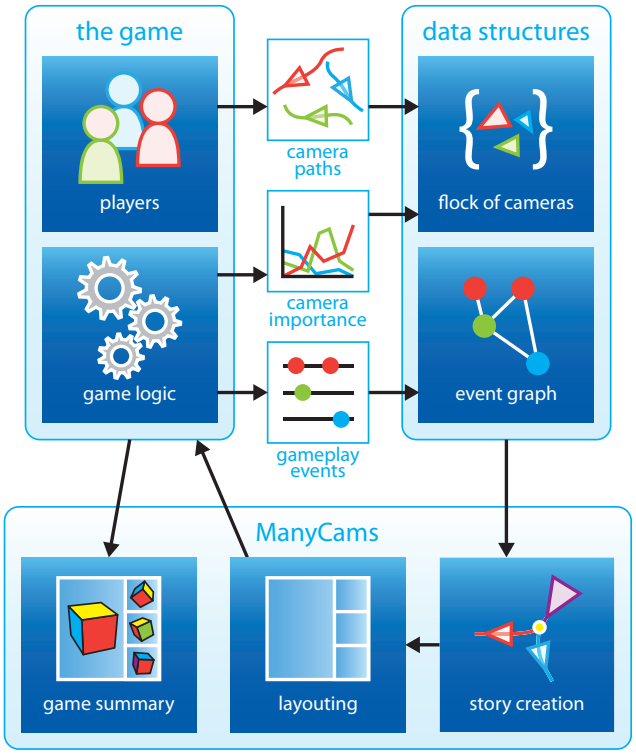


Figure 1: Overview of the ManyCams, the flock of cameras, and the data on which they operate. Camera paths, camera importance, and the events of the gameplay are extracted from the game and stored in the flock of cameras and the event graph. These structures are then processed by our ManyCams method to produce the video summarizing the gameplay.

multiple virtual cameras to simulate real-world problems, such as person-tracking in video-surveillance feeds.

In this work, our goal is to display 3D scenes in such a way that multiple dynamic foci are captured in a clear and semantically meaningful way. For this purpose, we utilize a flock of cameras, which is composed of multiple camera paths. The paths of the cameras can be either directly extracted from the video game, such as view paths of individual players, or created by game-level designers. Each camera path has a specific lifespan and it is assigned a continuous importance function over the period of its existence. The focus of this paper is not on creating these camera paths, but instead lies in visualizing image outputs of the cameras according to their importance. The intent is that the captured events are meaningfully linked together and the gameplay can be easily analyzed. The goals are: to display outputs of the cameras with high importance, to make the changes between individual cameras smooth and continuous so that users can easily maintain an overview, and to simplify the flock of cameras in a smart way so that the method can be effectively used for summarizing and analyzing gameplays. Our proposed method extends existing storytelling approaches by creating animations where multiple foci, or game events, possibly overlapping in time or space, are conveyed by the generated visual story.

3 Overview

We solve the problem of composing gameplay summary-videos from available gaming data. Figure 1 shows an overview of our method. The black arrows illustrate the data flow. First, camera paths are extracted from the game, e.g., views of individual players as they move through the game world. Additionally, the importance of the individual views in time as well as interesting game events are extracted during the gameplay. Our method does not assume particular event types or functions which evaluate the importance of each view. This information has to be provided by the game, since it depends on the game logic. Our method is only concerned with building the video summary of the gameplay described by the extracted data. If the game logic does not inherently contain any metric which could be used to evaluate the importance of the individual cameras, an importance of individual event types could be specified manually instead.

As a tangible example, let us consider a deathmatch game (where the goal is to kill all opponents) between four players A, B, C, and D. During the game, all but one players are sequentially killed. The game ends as soon as there is just one last player alive. Suppose the gameplay progresses as follows: Players A and B join the game and meet at a certain place. Soon after A spots C and kills him. Player C has joined the game shortly before that. After C is killed, he drops his weapon. Meanwhile, D also joined the game. He finds the weapon of player C. He picks it up and uses it to kill A. Shortly after that he also kills player B. At this point the game ends, because D is the last player alive.

The goal of our method is to generate a video, which visually tells this story by showing sequences of what individual players saw during the gameplay. We achieve this by extracting the data about the players and the game events (e.g., a player kills another one) from the gameplay. The extracted data are organized in data structures which are further described in Sections 3.1 and 3.2. These structures, i.e., *flock of cameras* and *event graph*, are processed by *ManyCams* after the gameplay finishes. The ManyCams method integrates the cameras into several continuous views which visually express important events as well as their causal contiguity, forming a visual story. Afterwards, the layouting algorithm juxtaposes the generated views in space or time in the screen space, so that the game summary can be generated. This information is sent back to the game engine, which then renders the game world from the perspective of the views generated by the ManyCams method to produce the final gameplay summary-video.

3.1 Flock of Cameras

We assume that the events which occur during the gameplay are captured by the cameras present in the 3D scene, such as views of individual players, or cameras surveying certain locations. While the players control the cameras capturing their views by navigating their avatars, the surveying cameras can be either positioned manually by the level designers or their parameters can be determined dynamically by one of many camera-path planning-methods [Halper et al. 2001; Oskam et al. 2010; Yeh et al. 2011].

A flock of cameras is a structure which maintains all the necessary information about the cameras in the 3D scene. A flock of cameras $F = \{c_0(t), ..., c_n(t)\}$ is a finite set of camera functions $c_i(t)$. For every time step t, $c_i(t)$ specifies the camera parameters $p_i(t)$ as a transformation matrix, an image of a camera view $v_i(t)$ rendered using $p_i(t)$, and a camera importance $i_i(t)$ as a real number in the interval $[0,1]$, where 0 signifies the lowest importance.

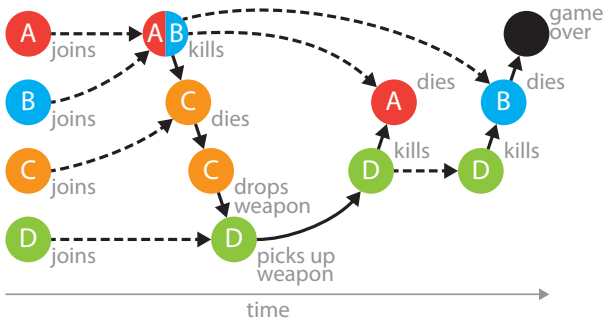


Figure 2: An example of an event graph. The circles indicate individual events, the solid arrows are causal links extracted from the game, and the dashed arrows are player-links derived from the flock of cameras.

3.2 Event Graph

The events occuring in the game are not separate entities, but rather interconnected elements of a narrative portraying the action of the gameplay. In our approach, an effective visual summary of the gameplay should convey the causality of the individual events. This way, the summary tells a coherent story, which can be followed in order to understand what happened during the gameplay. Understanding the story of the gameplay allows us not only to analyze the behaviour of the individual players, but also enables us to see the consequences of their actions.

To compose a meaningful narrative of the gameplay, we extract events and their causality from the game while it is played and store this information in an *event graph*. The types of events and their connections depend on the actual game. For instance, in a first person shooter, there are events such as *player A shoots* or *player B dies*. If player *A* killed player *B*, the event *player A shoots* caused the event *player B dies*. Another example is that player *A* throws a grenade and kills several other players. An event can cause zero, one, or several other events, and it can be caused by zero, one, or more events. We connect causally dependent events in the event graph by *causal links*.

Another aspect of gameplay dramatization in multiplayer games is that there are possibly multiple players participating in individual events. As they move through the game, individual players or groups of players participate in different events. Events with the same participants are semantically related within the story. To capture this concept in the event graph, we connect such events by *player-links*.

To build the player-links, we use the information stored in the flock of cameras. We treat each camera in the flock as a player. Therefore, all entities which are able to observe the action of the game, such as spectators or surveillance cameras, are treated as players. Instead of participating players, each event is associated with a set of cameras which observe the event. The player-links are then constructed based on the events' associations with the cameras from the flock. Details are provided in Section 4.1. Besides actual players, this abstraction also accounts for non-playing participants, such as spectators in first person shooters.

The causal links and the player-links connect the events according to the interactions of the players during the gameplay. Therefore, the event graph abstracts the story of the gameplay. Figure 2 shows the event graph of the previously described example gameplay. The circles represent the events of the game, while the letters indicate which players participated in individual events. The causal links (solid arrows) indicate how individual events are related according to the game logic. The player-links (dashed arrows) indicate how players moved between events. Both types of links encode the progression of the story which we want to tell.

The event graph is a directed graph G:

$$G = (E, K) \tag{1}$$

$$E = \{e_0, ..., e_n | e_i = \{a_i, t_i, d_i, P_i\}\} \tag{2}$$

$$K = \{k_0, ..., k_l, r_0, ..., r_m | k_i = (e_x, e_y) \in E^2, r_j = (e_z, e_w) \in E^2\} \tag{3}$$

E is the set of all events e_i which occurred during a gameplay. Each event consists of the event type a_i, the time of occurrence t_i, the duration of the event d_i, and a set of cameras P_i which are associated with the event.

K is the set of the edges representing links between the events. It consists of causal links $k_0, ..., k_l$ extracted from the game, and player-links $r_0, ..., r_m$ derived from the flock of cameras. Each link is an oriented edge between two events in the event graph.

3.3 Camera Operators

In order to produce a summary of the gameplay telling a story of what happened in the game, the events from the event graph need to be displayed using the available cameras. A naïve approach would be to sequentially show views from all the cameras stored in the flock. In this way, every event is included in the story, since everything that the game participants saw during the gameplay is shown. However, such a summary does not communicate any causality between the events, other than what each of the players did separately. In this case, the story of the gameplay is not effectively conveyed. An additional drawback is that possibly redundant and unimportant information is present.

Another approach, often employed in video surveillance, is to show all the views simultaneously. Although all the events captured by the cameras would be shown on screen, the viewer would potentially have to focus on multiple events at the same time. This compromises the utility of the approach for scenarios such as multiplayer video games, where multiple dynamic foci possibly occur simultaneously.

In our work we aim to overcome the drawbacks of the mentioned methods. We compose the visual story by following the links between events in the event graph, so that the coherence of the action's portrayal is maintained. Additionally, the redundancy in the views of the available cameras is reduced before the views are employed to display individual events. This way, the generated video summary tells a story which helps to make sense of the events in the game.

The method presented in this work operates on virtual cameras displaying 3D scenes. In contrast to real-world scenarios where multiple video cameras are employed, such as sports events or video surveillance, processing virtual cameras enables the modification of the camera parameters and rendering of the scene from new viewpoints. We utilize this possibility by applying various operators on the flock of cameras. They reduce redundant information and provide visual links between related events.

The *Overview* is an operator which shows the generated visual narrative from a bird's-eye view provided by a dedicated overview camera placed above the scene. The players, their movements, and

Figure 3: Some of the pictograms used by the overview operator for a schematic display of the gameplay.

actions are depicted with animated pictograms (some of them are shown in Figure 3). This operator is inspired by schematic depictions used in sports to analyze the strategies of a team.

The *Time-lapse* operator smoothly changes the speed of the generated visual narrative. It is used to seamlessly pass through uninteresting parts of the story. It avoids creating a hard cut, which might be confusing in a summary of a fast-paced video game.

The *Mark-player* operator is used to point out individual events shown by the generated visual narrative. It stops the playback when an important event occurred, and highlights the entity which caused the event. In our use case, we apply this operator to mark players important for the story. The operator is based on the practice often employed in live-tv broadcasting of sports events, when the broadcast video is briefly stopped so that an important object or person can be marked to guide the viewers' attention.

The *Camera-merge/split* operators are applied to multiple cameras showing the same event from similar viewpoints. If several players move together as a group in the same direction, the camera-merge operator is applied. When they no longer move together, the camera-split operator is used. When the cameras are merged, their views can be used in the summary video interchangably. The merged cameras are seen as a single entity within the story.

The *View-switch* operator is applied if events occurring at different times or places need to be shown in a sequence. Instead of jumping from one event to the other one, the operator interpolates between the camera parameters of the views capturing both events. Simultaneously, time is interpolated from the end time of the first event to the beginning time of the second event. If the second event occurred before the first one, the time flows backwards during the interpolation, giving the view-switch a dramatic effect. The view-switch communicates spatial (by interpolating camera parameters) and temporal (by interpolating time) relationships between the two events shown in a sequence. Similar techniques are often used in cinematography, where multiple scenes are displayed in one continuous shot to communicate the continuity of the portrayed events.

2D inset-show/hide operators show and hide 2D insets of a camera view which is currently not the main focus, but is relevant to the current situation. An example scenario is when several players observe the same situation from substantially different viewpoints which cannot be merged. Since these players observe the same focus, it is possible to display their views in parallel without confusing the viewer of the summary.

4 ManyCams: Summarizing the Gameplay

ManyCams is a method for traversing the event graph and matching patterns with operators appropriate for the given situation, which are then applied to the flock of cameras to produce a summarizing visual narrative of the gameplay. To achieve this goal, it is necessary to create a virtual representation of the story, and to provide an algorithm for creating a visual narrative from this representation.

4.1 Story Representation

The story is represented by the event graph. The data extracted from the game contain the nodes of the graph (events), and causal links between them. In order to represent the story of the gameplay by the event graph, it has to be further pre-processed.

First, the events are associated with the cameras from the flock. Depending on the game, this information might be available from the extracted gaming data. Alternatively, we provide an implementation of a density-based clustering algorithm DBSCAN [Ester et al. 1996], which can be used to associate multiple cameras with similar views with a single event. However, a heuristic for estimating the similarity between camera views has to be provided as well. For first-person shooter games, we use the sum of spatial-position differences and viewing-angle differences as the heuristic. This can be evaluated quickly, while it is robust enough to efficiently cluster views of players in first-person shooter games.

The duration of the summary video can be specified. Only the n most important events, which would fit into the specified duration, are included in the summary video. Since the events are associated with the cameras from the flock, the time-varying importance of the cameras can be used to automatically estimate the importance of the events. If there is no camera importance information available in the extracted gaming data, the importance of individual event types can be specified manually.

Finally, player-links are added to the event graph. A player link is created between each pair of chronologically successive events (e_i, e_j) with sets of associated cameras P_i, P_j, such that $P_i \cap P_j \neq \emptyset$. If there are multiple events fulfilling this condition for the event e_i, only the one with the closest timestamp is chosen to create the player link.

4.2 Building the Visual Narrative of the Gameplay

The summary video is constructed by appliyng operators on the flock of cameras. In the story of the gameplay scenario described at the beginning of Section 3, players A and B meet and for a period of time, they observe similar parts of the scene. Here, the camera merge operator can be applied. At times, the described narrative is non-linear in the sense that the events are not described in their chronological order. In the visual summary, this could be achieved by applying the view-switch operator. The story itself follows the links in the event graph. By traversing the event graph and applying the camera operators, it is possible to generate a visual summary narrating the story as described in the beginning of Section 3.

To create the summary video of the gameplay, the layouting algorithm arranges the camera views of the events, while it applies the described camera operators. It begins by showing the view of the chronologically first event. The algorithm continues by traversing the event graph in a depth-first order, following the causal links and player-links. In this way, the story progresses by placing semantically relevant events, as determined by the links successively connecting them. The links to the events with higher importance are followed first. When the next event is shown, the camera merge operator is applied to all cameras associated with it, and the camera split operator is applied when the event finishes. This ensures that the events with multiple participating players are not shown multiple times. The depth-first search of the event graph ensures that showing related events in sequence is preferred to applying the view-switch operator between unrelated events. This keeps the coherency of the parallel narrative of the visual story.

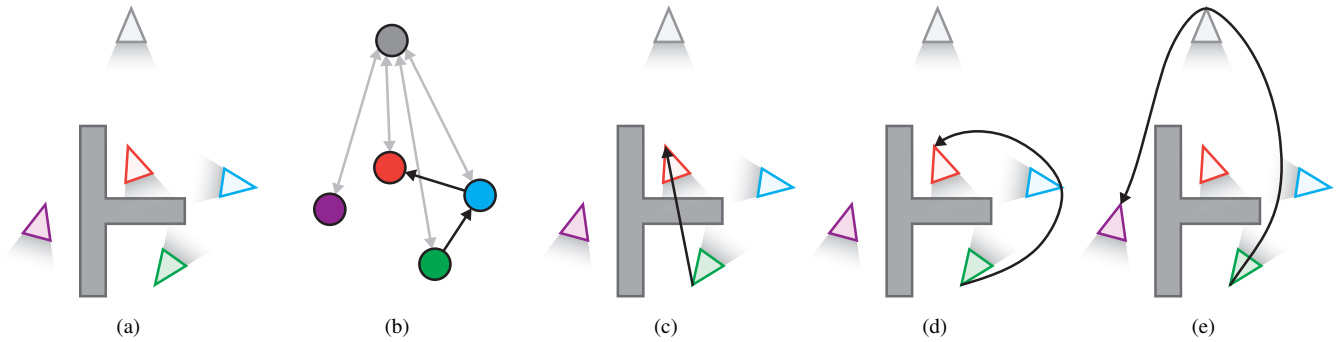


Figure 4: (a) Five cameras in a scene. The gray one is an overview camera. (b) A visibility graph. (c) Switching view from the green camera to the red one would cause a collision with a wall. (d) The view from the green camera can be switched to the view from the red camera through the blue camera, because such a path exists in the visibility graph. (e) The views from any pair of cameras can be switched through the overview camera (as indicated by the gray arrows in (b)).

4.2.1 View-Switch Operator

In our storytelling approach employing multiple cameras capturing the events of the gameplay, it is often necessary to switch between views. Our method tries to sequence the events so that the ones which are temporally adjacent in the summary video are related, as indicated by the edges in the event graph. Since the event graph is not necessarily traceable (i.e., containing a Hamiltonian path), this might not be possible for the whole story. Therefore, we provide the view-switch operator which allows us to seamlessly switch between views of potentially unrelated events.

During the rendering of each frame, we build a *visibility graph*. It is a directed graph, where the nodes represent individual cameras from the flock, and the directed edges from a node indicate which cameras are seen by the camera represented by this node. Figure 4a illustrates a set of cameras and Figure 4b shows their corresponding visibility graph. The overview camera, which sees all the players, is included in the calculation of the visibility graph. It ensures that a path exists between any two cameras in the flock. For the overview camera, the rendering has to be adjusted, e.g., by making ceilings transparent, so that all the players are visible.

We use the visibility graph for seamless view switching. If the view from a camera should be switched to the view of another camera, we first find the shortest path between nodes in the visibility graph the Dijkstra algorithm [Dijkstra 1959]. Subsequently, we create a curve spatially connecting cameras of all the views on the path. This curve can be used to smoothly interpolate between the desired views. This approach ensures that the generated curve does not intersect any obstacles in the 3D scene, since the cameras on the path are sequentially visible from each other. Figure 4c illustrates an undesired situation if two cameras would be interpolated directly. This would result in collisions with the scene geometry. Our approach employing the visibility graph is illustrated in Figure 4d.

The view-switch operator seamlessly interpolates between views of two events. Since the time of the occurrence of these events might not correspond with the length of the view-switch, the current timestep of the gameplay is interpolated together with the camera parameters. This means the time might be dilated during the view-switch. It might even flow backwards, in case the events occurred in reversed order compared to how they are shown in the summary video. The effect of reversing the time while moving the camera to a new location is used in cinematography if it is important to correctly portray the order of the events.

4.2.2 Overview Operator

The player-links in the event graph join events where the same players are participating. Since the story is constructed by following these links, it can be split into continuous blocks of events involving individual players. Before each block, we apply the overview operator, which shows a schematic depiction of movements and actions of the involved players from the perspective of the overview camera. Each event is illustrated by a simple animation, while the movement of the players between temporally adjacent events is shown by low-pass filtered paths of their positions recorded during the game.

After the whole event sequence is shown in the overview, we apply the view-switch operator to rewind the time and show the same sequence of events from the viewpoint of the player associated with the events in this story block. If there are several players associated with all the events in the block, any one of them is chosen, since their views are sufficiently similar.

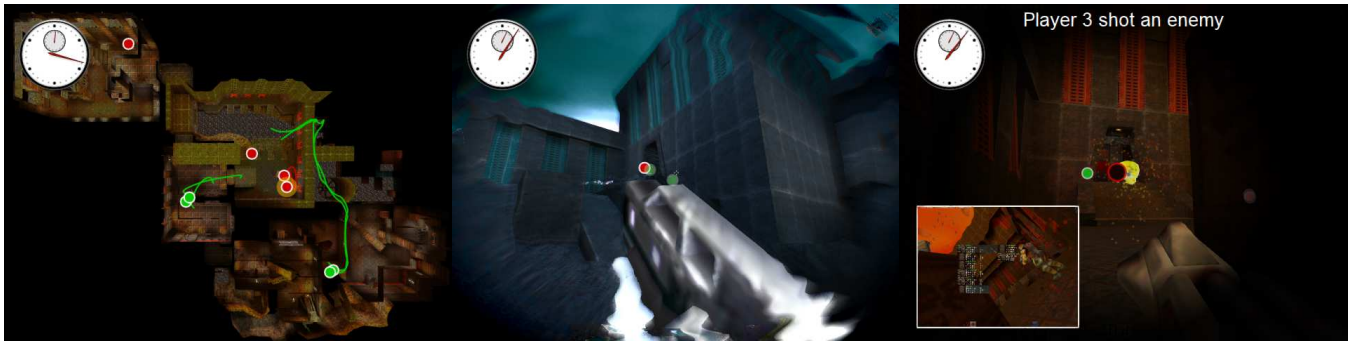
This combination of operators gives the audience an initial overview, followed by a detailed depiction of the events, as seen by the players. The view-switch operator provides a spatial and temporal reference to the other parts of the story, since the camera always moves continuously in space and time.

4.2.3 2D Inset-Show/Hide Operator

2D inset operators are used in situations where it is beneficial to show multiple camera views juxtaposed in image space. If a currently shown event caused an event associated with another player, the view of the other player is shown in the 2D inset. For instance, when player A kills player B, we show the view of the player A on the entire screen, while the view of player B is shown in the 2D inset. The same situation is shown from two different perspectives.

4.2.4 Camera Operators for Enhancing First-Person Views

If the view of a player moving between related events is shown, the time-lapse operator continuously increases the playback speed, as long as no important events are in the depicted field of view. As the player is getting closer to a certain event, the playback is continuously slowed down. This form of temporal fish-eye lens allows us to connect related, but rather distant events, by a continuous view



(a) (b) (c)

Figure 5: A summary video of a gameplay of Jake 2 generated by ManyCams. (a) Overview operator showing the movements of two groups of players from the green team, trying to surround players of the red team. (b) View-switch operator continuously changing view towards an event which has already occured. To achieve this, time is reversed, which is communicated through a distinct visual style applied to the entire screen. (c) The mark-player operator (red circle in the middle) indicating a player of the opposing team being shot. Description of the event is shown at the top. 2D inset in the left bottom corner shows the view of the dead player. The surrounding is darkened to guide the viewer's attention towards the event.

of the player. In case either the view-switch or time-lapse operator changes the playback speed, a distinct visual style is applied to illustrate the modified time flow (Figure 5b).

It is important that the changes in the playback speed are smooth so that they are not distracting the viewers. For this reason, we resample the individual frames in the time domain. Slowing the playback down is achieved by interpolating between successive frames. Making the playback faster is done by averaging several frames. A drawback of both approaches is the introduction of a certain blur.

If another player interacts with the player whose view is currently shown, causing an event (such as shooting), the *mark-player* operator is applied. The time stops for a brief moment, and the position of the interacting player is highlighted with a growing circle. The circle is color-coded according to the type of caused event (e.g., shooting or spotting). At the same time, the surroundings of the marked player are darkened, to further guide the attention of the viewers, as depicted in Figure 5c.

5 Results

In order to evaluate our method, we apply it to a real computer game. We use the multiplayer, first-person shooter game *Jake2* [Bytonic Software 2006], a Java port of *Quake II* by *id Software*, to acquire the gaming data consisting of movements and actions of the players, their views, and game events. These data are processed by our method, which calculates additional views used for smooth transitions between the shown events, and composes the summary video. It also inserts a stopwatch into the generated video which helps to indicate the flow of time. Text captions describing the currently shown events from the event graph are automatically inserted as well. Still images from a generated summary video are depicted in Figure 5. The full video is included as supplementary material.

Our method is mainly targeted for games played in teams, which often utilize various strategies. Our method is particularly useful for reviewing the execution of such strategies, either for training or entertainment purposes. We recorded several *team deathmatch* games, where the goal is to eliminate all players of the opposing team. There were two teams, each of four players. The players were given strategies, which they followed during the game. The summary video of the game generated by our method clearly shows these strategies and tasks carried out by individual players. We showed the summary video to experienced video-game players. They confirmed that the video sufficiently illustrates the performed strategies and tasks of individual players, even though at some instances it was not immediately clear to them why the events were shown in a particular order.

6 Discussion

Our gameplay-summarizing method captures multiple temporarily and spatially overlapping events in a visual story with a parallel structure, which narrates the course of the gameplay. We organize multiple cameras capturing the gameplay into a flock of cameras, and store game events in an event graph. The redundancy in the views shown by the cameras of the flock is reduced by applying several operators. The story is constructed by displaying the camera views capturing the most important events. Since the action in multiplayer games is often complex, smooth transitions are placed between the displayed events so that their spatial relations and temporal succession is communicated. Our method allows users to analyze the behaviour and interactions of participating teams or individual players during the gameplay.

7 Conclusions

In this paper, we described a system for generating summary videos of multiplayer gameplays. These summaries are suited for entertainment, or for analyzing and exercising team strategies. The challenge of this approach is to deal with multiple players, who interact with the system and create multiple, possibly concurrent events at different places. Our method smoothly integrates the recorded gaming data of all players, utilizing overviews and time dilation effects, so that the gameplay is illustrated as a continuous and expressive visual story.

Acknowledgements

The presented work has been partially supported by the ViMaL project (FWF - Austrian Research Fund, no. P21695), Vienna Science and Technology Fund (WWTF) through project VRG11-010, and by the Aktion OE/CZ grant number 68p5. We would like to thank all the participants of the conducted gaming sessions.

References

AGRAWALA, M., ZORIN, D., AND MUNZNER, T. 2000. Artistic Multiprojection Rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, Springer-Verlag, London, UK, 125–136.

BARNES, C., GOLDMAN, D. B., SHECHTMAN, E., AND FINKELSTEIN, A. 2010. Video tapestries with continuous temporal zoom. *ACM Transactions on Graphics (Proc. SIGGRAPH) 29*, 3.

BERKELEY, L. 2006. Situating machinima in the new mediascape. *Australian Journal of Emerging Technologies and Society 4*, 2, 65–80.

BYTONIC SOFTWARE, 2006. Jake2. `http://bytonic.de/html/jake2.html/`. Accessed: 2015-3-17.

CHEONG, Y.-G., JHALA, A., BAE, B.-C., AND YOUNG, R. M. 2008. Automatically generating summary visualizations from game logs. In *AIIDE*, The AAAI Press, C. Darken and M. Mateas, Eds.

CHRISTIE, M., OLIVIER, P., AND NORMAND, J.-M. 2008. Camera control in computer graphics. *Computer Graphics Forum 27*, 8, 2197–2218.

CORREA, C. D., AND MA, K.-L. 2010. Dynamic video narratives. *ACM Transactions on Graphics (Proc. SIGGRAPH) 29*, 3.

DIJKSTRA, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

ESTER, M., KRIEGEL, H. P., SANDER, J., AND XU, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Portland, Oregon, E. Simoudis, J. Han, and U. Fayyad, Eds., 226–231.

GERSHON, N., AND PAGE, W. 2001. What storytelling can do for information visualization. *Commun. ACM 44*, 8 (Aug.), 31–37.

HALPER, N., AND MASUCH, M. 2003. Action summary for computer games – extracting and capturing action for spectator modes and summaries. In *Proceedings of 2nd International Conference on Application and Development of Computer Games*, Division of Computer Studies of the City University of Hong Kong, Hong Kong, China, L. W. Sing, W. H. Man, and W. Wai, Eds., 124–132.

HALPER, N., HELBING, R., AND STROTHOTTE, T. 2001. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum 20*, 3, 174–183.

HE, L.-W., COHEN, M. F., AND SALESIN, D. H. 1996. 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*, ACM, New York, NY, USA, SIGGRAPH '96, 217–224.

HORNUNG, A., LAKEMEYER, G., AND TROGEMANN, G. 2003. An autonomous real-time camera agent for interactive narratives and games. In *Intelligent Virtual Agents*, T. Rist, R. Aylett, D. Ballin, and J. Rickel, Eds., vol. 2792 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 236–243.

HSU, W.-H., MA, K.-L., AND CORREA, C. 2011. A rendering framework for multi-scale views of 3d models. *ACM Transactions on Graphics 30*, 6 (December).

LÖFFELMANN, H., AND GRÖLLER, E. 1996. Ray tracing with extended cameras. *Journal of Visualization and Computer Animation 7*, 4, 211–227.

MA, K.-L., LIAO, I., FRAZIER, J., HAUSER, H., AND KOSTIS, H.-N. 2012. Scientific storytelling using visualization. *Computer Graphics and Applications, IEEE 32*, 1 (Jan.–Feb.), 12 – 19.

OSKAM, T., SUMNER, R. W., THUEREY, N., AND GROSS, M. 2010. Visibility transition planning for dynamic camera control. In *Proceedings of the Third international conference on Motion in games*, Springer-Verlag, Berlin, Heidelberg, MIG'10, 325–325.

QURESHI, F., AND TERZOPOULOS, D. 2006. Virtual vision and smart camera networks. *Working Notes of the International Workshop on Distributed Smart Cameras (DSC 2006)*, 62–66.

SEGEL, E., AND HEER, J. 2010. Narrative visualization: Telling stories with data. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (Nov.), 1139–1148.

TWITCH INTERACTIVE, INC., 2011. Twitch. `http://www.twitch.tv/`. Accessed: 2014-2-21.

VIOLA, I., FEIXAS, M., SBERT, M., AND GRÖLLER, M. E. 2006. Importance-driven focus of attention. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (Oct.), 933–940.

WOHLFART, M., AND HAUSER, H. 2007. Story telling for presentation in volume visualization. In *Proceedings of the 9th Joint Eurographics / IEEE VGTC conference on Visualization*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EUROVIS'07, 91–98.

YEH, I.-C., LIN, C.-H., CHIEN, H.-J., AND LEE, T.-Y. 2011. Efficient camera path planning algorithm for human motion overview. *Computer Animation and Virtual Worlds 22*, 2-3 (Apr.), 239–250.

YU, L., LU, A., RIBARSKY, W., AND CHEN, W. 2010. Automatic animation for time-varying data visualization. *Computer Graphics Forum 29*, 7, 2271–2280.