Toward the Automatic Identification and Generation of Highlight Cinematics for 3D Games

Michael Dominguez, Stephen Roller, and R. Michael Young
Liquid Narrative Group, Department of Computer Science, North Carolina State University
E-mail: {madoming, scroller, young}@ncsu.edu

Abstract
Online multiplayer gaming has emerged as a popular form of entertainment. During these games, the players' main focus is usually placed on achieving the objectives that must be completed to win the game. Over the course of the game, however, their interactions may result in interesting emergent narratives that go unnoticed. Afterthought is a system that monitors player activity, recognizes instances of story elements in gameplay and renders cinematic highlights of the story-oriented game play, allowing players to view these emergent narratives after completing their gameplay session.

Introduction
Over the course of an online multiplayer gameplay session, interesting narratives can emerge through the interactions between players. In many cases, these may go unnoticed by the participants, possibly due to the players’ primary focus on the completion of the game’s objectives or their lack of complete knowledge of the dynamic state of the virtual world. In order to experience these narratives, players need a method of reviewing past actions that took place during gameplay.

Many commercial games provide methods to analyze past gameplay. A typical player-controlled replay system gives users free control of the viewpoint, putting the burden on the player to find the most interesting moments and to position the camera such that they can adequately see the action. Other games allow players to view instant replays of recent gameplay. These may be shown automatically by the game after it recognizes an important occurrence, or in response to requests made by players mid-game.

While these existing systems provide many useful features, they lack the ability to both intelligently recognize complex interactions between players during gameplay and effectively create a video illustrating them across a range of narrative and geometric contexts. Related research includes work done by Cheong et al. (2008), who developed a system called ViGLS (Visualization of Game Log Summaries) that aimed to automatically create video summaries of past gameplay. Friedman et al. (2004) also worked on creating video summarizations based on logs of activity in virtual worlds. Chan et al. (2009) describe a mixed-initiative system used to generate comic panels summarizing gameplay. A earlier comic generation system was created by Shamir, Rubinstein, and Levinboim (2006).

Description
Afterthought is a system that monitors player activity, recognizes instances of story elements in gameplay and renders cinematic highlights of those elements. There are five components of Afterthought: the modified game to be used with the system, the log dispatcher, the narrative pattern matcher, the cinematic discourse generator, and the video renderer. As a multiplayer game takes place, player actions trigger log messages that are sent from the modified game to the log dispatcher. The log dispatcher forwards the messages to the Narrative Pattern Matcher (NPM) and to the Cinematic Discourse Generator (CDG). As log messages are received, the NPM identifies every sequence of gameplay actions that match the narrative fragments that it has been built to detect. At the conclusion of the game match, the NPM sends these narratives to the Cinematic Discourse Generator. The CDG then determines which narratives to film and how they should be shot and edited. Instructions for filming are sent to the Video Renderer, which records the videos and uploads them to a video sharing web site. This system has been fully implemented for the Capture the Flag game type in the first-person shooter Unreal Tournament 3.

Game Server and Log Dispatcher
The host of a multiplayer game acts as the game server. When the game begins, the server establishes a socket connection to the Log Dispatcher. As important player actions take place in the game, the server sends log messages, encoded in XML, via this connection to the Log Dispatcher. The Log Dispatcher forwards these log messages to the NPM and the CDG. When the game session has completed, the end game log message instructs the NPM that it will receive no more actions and that it can begin sending matched narratives to the CDG.
Narrative Pattern Matcher

Before the game begins, a user defines the narrative patterns the NPM should look for. Each pattern consists of a number of specific actions composed with regular expression-like operators. When the NPM starts, patterns are converted into Finite State Machines (FSMs). Variables in patterns ensure distinct actions within a pattern share common elements, such as a player’s name or team color. Furthermore, pattern constraints are added by performing extra checks in an FSM’s transitions. Constraints may include timing restrictions, explicitly disallowed actions and comparison relationships between variables.

After creating FSMs for each pattern, the NPM starts receiving messages from the Log Dispatcher. Each pattern receives its own match tree that keeps tracks possible matches. Each node in the match tree corresponds to a complete state of the FSM. Incoming messages are checked against each node in the match tree; if a message causes a state change in the FSM, a new branch is created. After processing all log messages, an XML document containing the matched narratives is passed to the CDG.

Cinematic Discourse Generator

During the gameplay session, the CDG receives the log messages from the log dispatcher. After the gameplay session is complete, it receives a collection of the game actions that have matched any narrative patterns from the NPM. After receiving the narratives, it must determine how to film each narrative. Each narrative is comprised of a sequence of actions that took place during the gameplay session. For a particular narrative, the output of the CDG is a camera event containing the full specifications that the Video Renderer needs to film the narrative.

To begin the construction of a camera event, the first step is to load the camera action series that is associated with each action in the narrative. A camera action series defines the optimal way of filming an action. An action may not be filmed as described in its camera action series because, in the generation of the camerawork for an entire narrative, certain elements may have to be sacrificed in order to effectively create the entire video.

A camera action contains the details for the specifications for the camera’s degrees of freedom (position, rotation, field of view) over a certain period of time. The camera actions in a camera action series are defined as either primary (one per camera action series) or secondary. A primary camera action will be in effect at the moment that the camera action series’ associated action occurs during the replay. The secondary camera actions, which can occur before and/or after a primary camera action, may be removed due to timing conflicts when attempting to construct a camera event to film the entire narrative. A primary camera action is never removed due to timing conflicts; if a timing conflict exists, its duration is shortened and a slow-motion effect is applied. This has the added benefit of providing emphasis to the important actions being performed by the players.

Video Renderer

XML camera specifications for recording all the narratives are sent from the CDG to the Video Renderer. The renderer instructs playback of the gameplay session to begin. As the game begins playback, it requests the camera event from the renderer. The camera actions in the camera event are applied during playback so that the appropriate camera shots are used to capture the actions that take place in the game. The renderer utilizes Fraps, a video capture utility, to record the game’s viewpoint.

This process completes by creating a text file containing an AviSynth script that consists of commands to concatenate all the video clips depicting the narrative. After the AviSynth script is generated, VirtualDub, a video processing utility, concatenates the clips and compresses the movie to a smaller file size. After the compression is complete, the generated video is uploaded to a video sharing web site. After the completing the upload, the process begins for the next narrative and continues until finishing the recording of all narratives.

Future Work

We are currently in the process of conducting an empirical evaluation of the videos produced by Afterthought, measuring their quality as perceived by game players along dimensions that range from accurate depiction of game play to effective identification of story-like action sequences to the inclusion of surprising or humorous narrative elements.

From a technical perspective, there are many opportunities to improve upon our completed work. For instance, Afterthought has difficulty filming simultaneously occurring events. We hope to add support for multiple concurrent viewpoints in the generated videos. We also plan to increase the expressivity of the NPM so that more complex patterns can be located. In addition, we hope to extend Afterthought to other games and game types.

References