# Design and Evaluation of Afterthought, A System that Automatically Creates Highlight Cinematics for 3D Games

**Mike Dominguez**
FactSet Research Systems
601 Merritt 7
Norwalk, CT 06851
mdominguez@factset.com

**R. Michael Young**
Department of Computer Science
NC State University
Raleigh, NC, 27695-8206
young@csc.ncsu.edu

**Stephen Roller**
Department of Computer Science
University of Texas
Austin, TX
roller@cs.utexas.edu

**Abstract**

Online multiplayer gaming has emerged as a popular form of entertainment. the course of a multiplayer game, playerinteractions 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. This paper describes Afterthought's implementation as well as an empirical human-subjects evaluation of the effectiveness of the cinematics that it creates.

## Introduction

Over the course of an online multiplayer gameplay session, interesting narratives can emerge through the interactions between players. In many cases, these narratives may go unnoticed by the participants, possibly due to the players' primary focus on completing 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 moments of 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 taking place. Some replay systems give users the ability to pause, rewind, and fast forward through the action. Others include more advanced features such as clipping a replay to a certain segment and the integration of a mechanism to share clips with other players. Some games can automatically generate a video showcasing a highlight, however they may be limited to featuring only the most obviously important moments, such as capturing a flag or scoring a goal. While these existing systems provide many useful features, they lack the ability to both intelligently recognize complex interactions between players during gameplay and subsequently effectively create a video illustrating them.

For example, perhaps the following scenario occurs in a first person shooter. Player A enters a courtyard and is killed by Player B, a sniper in a bell tower. Player A respawns and enters the courtyard again, but is killed by a grenade thrown by his teammate, Player C. Player A respawns and enters the courtyard once more, and is killed by Player D's shotgun. Players B, C, and D may not be aware of Player A's repeated misfortune and could find this series of events humorous. An effective way of presenting these events to the players would be to create a video that shows shot sequences of Player B killing Player A, followed by Player C killing player A, followed by Player D killing Player A.

When a viewer experiences a visual narrative, the way that the actions of the story are presented to them can dramatically impact their perception and understanding of events. This paper describes an evaluation of *Afterthought*, a system designed to create videos that depict emergent narratives that occur during gameplay. The problems addressed in the creation of this system include a) the logging all of the significant actions that take place in a gameplay session, b) recognizing the narratives that occur in game logs, c) determining the most effective way of filming these narratives, and d) rendering the video file.

## Related Work

A number of existing commercial games perform analysis to identify exceptional action sequences and report them to players, for example by providing medals or achievements for killing an opponent that just killed a team mate (e.g., *Call of Duty: Black Ops*). Many games already include features that allow players to review their past gameplay sessions. However, these systems do not reason about the filming of complex interactions between players.

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 a player may choose to pause their gameplay in order to review some event (e.g., instant replays in *Mario Strikers Charged* or the "KillCam" in the *Call of Duty* series). However, these systems show only event sequences of very limited spatial or temporal scope and with very limited number and type of event, limiting their narrative generation abilities.

A number of systems have been developed to analyze the logs of gameplay for both virtual and live-action games.

Cheong et. al. (2008). developed a system called ViGLS (Visualization of Game Log Summaries) that automatically creates video summaries of past gameplay Friedman et. al. (2004) also worked on creating video summarizations based on logs of activity in virtual environments. In their approach, the actions in a log are compared against predefined interesting actions, characters, and series of actions. Halper and Masuch (2003) created a system that can extract and evaluate action scenes in computer games. They introduced an evaluation function used to determine the interest level at a particular time during gameplay.

## Description of Afterthought

### Overview

Afterthought (initially described in brief by the authors (Citation redacted for anonymity)), consists of five components - the modified game to be used with the system, the log dispatcher, the narrative pattern matcher, the cinematic discourse generator, and the video renderer. A host system initializing the multiplayer match acts as the server for the game session. The systems of the other players in the match connect to the server as clients prior to the beginning of the game. The log dispatcher also connects to the game server over a socket. As the game is being played, actions performed by the players trigger log messages that are sent from the game server to the log dispatcher. The log dispatcher then forwards the messages to the narrative pattern matcher and to the cinematic discourse generator. The narrative pattern matcher finds every series of gameplay actions that match the narrative patterns it has been provided. At the conclusion of the game match, the narrative pattern matcher sends these sequences, called *narratives*, to the cinematic discourse generator. The cinematic discourse generator then selects one or more narratives from this collection to film and determines how they should be shot and edited. The resulting instructions for filming are then sent to the video renderer, which replays the game session, records the videos and uploads them to a video-sharing web site.

For this work, a mod that provided for integration with Afterthought was created using *Unreal Tournament 3* (UT3). The evaluation of the work, described below, makes use of the UT3 Capture the Flag game type.

### Game Server and Log Dispatcher

A host system of a multiplayer game acts as the Afterthought UT3 game server. When the game session begins, the server establishes a socket connection to the log dispatcher. As important actions take place in the game, the server sends log messages describing these actions, encoded in XML, out over the connection. See Table 1 for an enumeration of the action types logged by the system for UT3's Capture the Flag game type.

Table 1: All action types logged in the modification to *UT3*'s Capture the Flag game type.

| Message Types | | |
|---|---|---|
| Spawn | Begin Fire | End Fire |
| Kill | Damage | Switched Weapon |
| Pickup Item | Flag Score | Flag Pickup |
| Flag Drop | Flag Return | Sees Player |
| Stops Seeing Player | Enter Zone | Leave Zone |

### Narrative Pattern Matcher

When the log dispatcher receives messages from the game server, it forwards them to the narrative pattern matcher. The narrative pattern matcher finds every series of actions that match one or more of a set of predefined narrative patterns. Before the game begins, a user defines the narrative patterns to be recognized by the pattern matcher. Each pattern consists of a number of specific actions composed with regular expression-like operators. Each input pattern is converted into a finite state machines (FSM). 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 enforced in the matching process by performing extra checks during transitions from one state to another. Constraints may include timing restrictions, explicitly disallowed actions and comparison relationships between variables.

A single action in a pattern is specified by listing the action type followed by the parameters to be bound. For example, `(kill :killerName ?KILLER :victimName ?VICTIM)` matches the kill action with the killer's name and the victim's name bound to variables. `kill` is the type of action, `killerName` and `victimName` are two of the values given with each `kill` log message, and these will be stored in `?KILLER` and `?VICTIM`. Users can also append metadata to each action definition using the + operator. The only metadata currently functional is `+focus`, which can be optionally specified to provide the cinematic discourse generator with guidance as to which object bound to a specific variable in the pattern is of primary importance in the matched action (this is typically used to indicate the player or NPC performing the action).

A series of patterns can be combined to form new patterns as follows:

- `(concat pat1, pat2, ...)` - Defines a pattern in which `pat1` is matched, then `pat2` is matched, etc.
- `(oneormore pat)` - Defines a pattern in which `pat` is matched one or more times.
- `(star pat)` - Defines a pattern in which `pat` is matched zero or more times.
- `(union pat1, pat2, ...)` - Defines a pattern in which `pat1` matches exclusively, or `pat2` matches exclusively, etc.
- `(repeat n pat)` - Defines a pattern equivalent to `(concat pat, pat, pat, pat, ...)`, with `pat` repeating `n` times.

Constraints can be enforced on patterns as follows:

- `(within-time T pat)` - Defines a pattern that machines `pat` only if all the actions in `pat` occur within `T` seconds.

- `(atleast-time T pat)` - Defines a pattern that matches `pat` only if the actions in `pat` occur over at least `T` seconds.
- `(dontmatch :bad_dfa badpat :good_dfa goodpat)` - Defines a pattern that matches `goodpat` only if `badpat` does not match over the course of `goodpat`'s occurrence.
- `(constrain C pat)` - Defines a pattern that matches `pat` if variable constraints `C` are satisfied.

Additional coreference constraints can be placed on variables within actions and constraints can be combined using the logical connectives *and* and *or*.

When the narrative pattern matcher receives the `endgame` log message indicating that the game match has completed, it sends to the cinematic discourse generator a set of *narratives* – collections of actions from the game play that match with a narrative pattern. Each narrative is a tuple $\prec A, P, B \succ$, where $A$ is a set of action bindings of the form $\prec \alpha_i, a_i, f_i \succ$. Here $\alpha_i$ is the identifier of the action that occurred in the game play log, $a_i$ is the identifier of the corresponding action in the matched pattern and $f$ is the (optional) focus specification indicating which variable in the action definition should be used as the camera's focus during filming. $P$ is the name of the matched pattern and $B$ is the set of variable bindings that were generated to match the patten to game play actions and their details.

## Cinematic Discourse Generator

After the gameplay session is complete, the cinematic discourse generator receives all narratives that have been identified by the narrative pattern matcher. For each narrative, the discourse generator computes a *shot list*, a set of directives indicating how the actions in the narrative are to be executed and filmed. The first step in the construction of a shot list is to load the *shot list* that is associated with each action $\alpha_i$ in the narrative. A shot list defines the optimal way of filming an action of a given type. An action is not guaranteed to be filmed as described in its shot list because in the generation of the camerawork for an entire narrative, certain elements may have to be sacrificed in order to effectively create a video. Each *shot directive*, or simply a *shot* in a shot list is defined with a file name, its timestamp and actor variables, and its constraints. The timestamp variables include the time that the shot starts and the time that the shot ends. In addition, a transition time – discussed below – may be specified. The position and rotation of the camera is defined relative to the actors specified in the actor variables. Each shot in the shot sequence is given a pre-defined, static importance rating; these ratings, running from 1 to 10, indicate the relative importance of the shot in the overall series. An importance rating of 10 indicates that the shot is a *primary shot*, meaning that it must be included in the final shot list that is generated. Each primary shot also has a must-include timestamp in its definition indicating a time point during which this shot must be active. Any shot with an importance rating less than 10 is considered a *secondary shot* and may be completely removed in the final shot list that is generated.

**Resolving Timing Conflicts Between Shots** Since only one shot may be active (or filming) at once, conflicts may exist among some of the starting and ending times for the shots specified in a shot sequence once their time-stamps are linked to the execution times of the actions they're filming. These conflicts need to be resolved before a video can be created. First, the timing conflicts regarding the primary shots are considered. The first step in resolving conflicts between primary shots is to check the spawn times for the actors associated with the shots. As the camera's position and rotation will be relative to the actors that they film, a shot should not begin until all of its associated actors have spawned.

Next, each primary shot is checked to see if it overlaps in time with the next primary shot in the shot sequence. If it does, then a compromise needs to be made to ensure that only one shot will be active at a time. To solve this, Afterthought checks the must-include timestamps for the shots and sets the ending time of the shot (and starting time of the next shot) to `(must-include timestamp for shot) + (must-include for next shot - must-include timestamp for shot) / 2`. However, the first shot is not allowed to extend beyond the ending point that had already been set for it.

Next, if the two consecutive shots have the same *primary target*, then we allow a *transition* to occur between the two shots. A shot's primary target is the actor for which the camera's position is relative. A transition refers to interpolating the position, rotation, and field of view from one shot to another. Using transitions as opposed to cuts when two consecutive shots have the same primary target allows for more dynamic camerawork and also aids in the prevention of cuts that would be disruptive to the viewer. Next, conflicts affecting the secondary shots are considered. As opposed to primary shots, secondary shots will not have any of their starting or ending times adjusted to resolve conflicts. Instead, if a conflict exists, they are completely removed. The first check for conflicts with secondary actions looks to see if any of the actors associated with the shot spawn between the starting time and ending time of the shot. If this is the case, then the shot is removed from the camera event. After removing a secondary shot, Afterthought checks to see if the removed action has an already-specified transition from it to another action. If it does, the transition is also removed. As primary shots may have already had transitions put in place to resolve their conflicts, we need to make sure not to supersede them.

Finally, we consider conflicts between secondary shots and other shots. When attempting to resolve these conflicts, shots are looped through in reverse order. This allows us to give preference to secondary shots that occur before the primary shot in their series. If the shot at the previous index has an ending time that is greater than the current shot's start time, then there is a conflict. If the shot at the previous index is a primary shot, then it has preference over secondary shots, so the current shot is removed. In addition, the shot at the index past the current shot has its transition time removed, if it is not a primary shot with a transition from another primary shot. If the shot at the previous index is not a primary shot, then we remove that shot and remove the transition time for the current shot. If we have removed any shot, then we check the same index again on the next iteration. Looping ends when a complete pass has been made over the shots without making any changes.

**Adding Slow Motion Effects** The next step is to add a slow motion effect to each of the primary shots. Adding slow motion to important shots allows the viewer more time to comprehend what is occurring and intensifies the action taking place (Arijon 1991). This is especially useful since trimming of primary shots to avoid timing conflicts may result in actions whose duration are very short.

**Adding Recording Actions** The final step in creating a shot list is to add the actions that are used to instruct the renderer when to start and stop capturing the game's viewport to a video file, as well as when the game should fast forward past gameplay that will not be filmed. The following guidelines are followed. Before the first shot in the shot list, a START CAPTURING action is added, with a starting time equal to the shot's starting time. Any time there is a gap between the ending time of one shot and the starting time of the next shot, a STOP CAPTURING action is added, with a starting time equal to the earlier shot's ending time, followed by a START CAPTURING action, with a starting time equal to the second shot's starting time. Before every START CAPTURING action, a FAST FORWARD action is added. The starting time is set at its preceding stop capturing action, or to 0.0 if it is being inserted before the first START CAPTURING action in the shot list. The ending time, indicating when the fast forwarding should cease, is set at the starting time of the START CAPTURING action. The FAST FORWARD action is used to skip past portions of gameplay that will not be included in the final recording. Fast forwarding is required for timely generation of replays using UT3 since the only means for replaying game play for recording requires re-running the entire game session. Other engines, however, may provide facilities to jump directly to world states in the middle of the game trace, in which case the FAST FORWARD action would implement this functionality. Finally, a COMPILE VIDEO action is added at the conclusion of the shot list, with a starting time equivalent to the starting time of the final STOP CAPTURING action. This action is used to instruct the renderer that all filming is complete for this narrative.

### The Renderer

The XML camera specifications for recording all the narratives are sent from the cinematic discourse generator to the renderer, which is the component responsible for rendering the videos of the narratives. The renderer instructs the game to begin playback of the gameplay session. As soon as the game begins playback, it requests the shot list from the renderer. After receiving it, the game, as instructed by the first step in the shot list, fast forwards to the timestamp where the first start-capture action needs to be executed. When a start-capture action executes, a message is sent from the game to the renderer telling it to begin capturing the game's viewport to a video file. The renderer utilizes Fraps, a video capture utility, to accomplish this. The subsequent shots are applied during playback so that the appropriate camera shots are used to capture the actions that take place in the game. When a STOP-CAPTURE action is executed, a message is sent from the game to the renderer telling it to stop capturing the game's viewport to a video file. The game then fast forwards again to the next start-capture to begin capturing the next sequence of actions in the narrative, and so on. The final action is a COMPILE-VIDEO action, which sends a message to the renderer that all video files have been captured for this narrative and it should now combine them into a single file.

This process begins by creating a text file containing an AviSynth (AviSynth ) script that consists of commands to concatenate all the video clips depicting the narrative. The AviSynth script describes edits for the resulting video so that the beginning of the video fades in from black and fades out to black at its conclusion. After the AviSynth script is generated, the file is opened by VirtualDub (VirtualDub ), a video processing utility. VirtualDub executes the instructions to concatenate the clips and also compresses the movie to a smaller file size. After the compression is complete, a utility based on an open source library (YouTube Uploader ) uploads the generated video to a YouTube account.

## Preliminary Evaluation

In our initial evaluation of Afterthought, we sought to determine the effectiveness of Afterthought's dynamically generated camerawork for creating a replay cinematic, in contrast to the standard third-person view used by many games. To do this, human subjects played short rounds of the Capture the Flag game type in UT3 while making use of the Afterthought system. Afterthought generated cinematics based on the subjects' gameplay, both using our custom camerawork and the default third-person view. Subjects were then presented with the videos to review. Subjects rated the videos on a number of qualities and indicated the degree to which they perceived their own character's actions to be featured events in the cinematic.

### Participants, Materials, and Apparatus

The participants consisted of 18 graduate and undergraduate students from North Carolina State University who took part in groups across three study sessions. Each subject received two reference sheets describing the keyboard and mouse controls needed to play as well as the rules for the Capture the Flag game type in *UT3*. Each Capture the Flag match was played on the map titled *Reflection*, which comes included with *UT3*. Modifications to the map and game type enabled our logging capabilities, however they did not affect gameplay. For the matches, the translocator item, which gives players the ability to teleport around the map, was disabled. Each game match lasted approximately 10 minutes.

### Procedure

At the beginning of a study session, each subject was assigned to a computer in a computer lab. They took a few minutes to review the reference sheets before playing a 5 minute practice Capture the Flag match. The purpose of the practice match was to allow participants to gain familiarity

Table 2: The standard deviation of the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative.

| Group | Exciting | Humorous | Dramatic | Interesting | Coherent |
|---|---|---|---|---|---|
| 30 pairings | 1.57 | 1.48 | 1.55 | 1.55 | 1.47 |
| 22 pairings | 1.59 | 1.32 | 1.51 | 1.57 | 1.55 |

Table 3: T-statistic for the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative.

| Group | Exciting | Humorous | Dramatic | Interesting | Coherent |
|---|---|---|---|---|---|
| 30 pairings | 0.93 | 2.23 | 3.07 | 0.24 | -0.12 |
| 22 pairings | 1.07 | 1.13 | 2.40 | 0.00 | -0.83 |

with the game. After the conclusion of this practice session, the participants took part in a Capture the Flag match during which the Afterthought system logged their actions. At the completion of this match, each participant then watched a replay of the game from the same first-person perspective that he or she had used while playing. As they watched the replay, they completed a survey which included a question asking them to choose three significant gameplay moments from their session. After all the participants completed this survey, they then watched four videos generated by Afterthought and completed a survey for each. This survey asked them to rate each video on a number of qualities (Exciting, Humorous, Dramatic, Interesting, Coherent) on a 7 point Likert scale. They also responded as to whether they witnessed or participated in the events depicted in the videos and whether any of the events were described in their answers to the first question on the previous survey (that is, were any of the events that the subjects had identified as significant game play included in the automatically generated videos). In addition, they were asked to provide their general thoughts regarding the video.

## Deciding What Narratives To Film

A large number of narratives may be sent from the narrative pattern matcher to the cinematic discourse generator, as it simply sends each and every narrative that matched a pattern over the course of a game session. However, it would be prohibitive to record a video for each narrative for our evaluation, so a method was devised to determine which narratives should have their videos rendered. First, each type of narrative was given a ranking based on pre-set values associated with the narrative patterns. The more desirable the type of narrative is to film, the higher rank it was assigned. The two narratives that occurred during a game session with the highest narrative type ranking (of different narrative types, ties broken by the number of shots in the narrative's generated camera event) were chosen to be filmed. Also generated were two videos that did not take advantage of several aspects of the system. These two videos filmed the same narratives as the previously mentioned videos. However, they did not have any slow-motion effects applied and they were shot entirely from the third-person camera view used in a typical video game replay. They were edited the same way as the previous two videos. To negate the effects of watching the videos in a certain order, viewing orders were counterbalanced across all subjects.

## Results

Due to technical difficulties, seven participants were not able to view the videos generated by Afterthought immediately following their viewing of the full replay. The responses for three of these participants were subsequently discarded for incomplete responses on their surveys. Results are presented both including and excluding the remaining four participants, who watched the videos and completed the surveys remotely later that day.

Of the 22 total responses to the question "When you were playing the game, did you witness or participate in any of the depicted events in the video?" 4 subjects responded *All*, 11 subjects responded *Some* and 7 subjects responded *None*. Of the 22 total responses to the question "Did you describe part of all of the events in this video in your gameplay survey?" 9 subjects responded *Yes* and 13 responded *No*.

In Tables 2 and 3, Type A indicates the set of generated videos using slow-motion effects and custom camerawork, while Type B indicates the set of generated videos that did not use slow-motion effects and only featured the default third-person camerawork. Each set contains both the 30 participant/narrative pairings, which includes all participants, and the 22 participant/narrative pairings, which removes the previously mentioned four participants.

The null hypothesis for this experiment was that the mean scores of the videos of Type A will be less than or equal to the mean scores of the videos of Type B in the five qualities being examined. The alternative hypothesis stated that the mean scores of the videos of Type A will be greater than the mean scores of the videos of Type B in the five qualities being examined. To find support for the alternative hypothesis, 1-sided t-tests were performed with a confidence interval of 0.01 (Bonferroni correction, 0.05/5). t-tests were employed in this case because the true standard deviation was not known, and the number of sample survey responses was small (less than 40).

In the 30 pairings group, with 29 degrees of freedom,

Table 4: The mean scores given to the specified qualities in each group.

| Group | Exciting | Humorous | Dramatic | Interesting | Coherent |
|---|---|---|---|---|---|
| 30 pairings, Type A | 4.97 | 2.57 | 4.87 | 4.83 | 4.77 |
| 30 pairings, Type B | 4.70 | 1.97 | 4.00 | 4.77 | 4.80 |
| 22 pairings, Type A | 5.00 | 2.27 | 5.05 | 4.77 | 4.59 |
| 22 pairings, Type B | 4.63 | 1.95 | 4.27 | 4.77 | 4.86 |

Table 5: The mean of the difference between the scores given by each participant for the video of Type A and the video of Type B of the same narrative.

| Group | Exciting | Humorous | Dramatic | Interesting | Coherent |
|---|---|---|---|---|---|
| 30 pairings | 0.27 | 0.60 | 0.87 | 0.07 | -0.03 |
| 22 pairings | 0.36 | 0.32 | 0.77 | 0.00 | -0.27 |

a critical value of 2.462 was used, while for the 22 pairings group, with 21 degrees of freedom, a critical value of 2.518 was used. These critical values were taken from a t-distribution table of critical values. Using these values with the data shown in Table 3, the null hypothesis was rejected in favor of the alternative hypothesis in the 30 pairings group in regards to the Dramatic quality. Also note the results of the questions asking if the subject witnessed/participated in the events depicted in the video and if the subject selected any of the moments in the video as significant on their Gameplay Survey. For these results, the four participants discussed earlier were not included.

## Discussion and Conclusions

The data supports that the participants found the videos of Type A more dramatic than the videos of Type B. It appears that the dynamic camerawork and slow-motion effects successfully added more tension to the generated highlight videos. Afterthought was able to find gameplay moments that some players had not witnessed or that they did not initially choose as moments on their Gameplay Survey. A number of insights were gained from the participants' answers to the short answer questions on the video surveys. One aspect of the videos that received multiple negative remarks was the use of close-up shots. Cinematographers often use a close-up shot to provide a strong sense of psychological identification with a character (Arijon 1991). Transitions from a medium or long shot of a subject to a close up is also employed to add tension to a scene. However, these benefits did not appear to translate to their use in the videos generated by Afterthought. One primary difference may be that in a live-action film, an actor's face can reveal emotional cues that are effectively conveyed through a close-up, while in the game, a character's model remains static throughout. Also, some participants remarked that they wanted to see the action taking place around the character rather than see a close-up shot.

The dislike of close-ups may also be related to another unfavorable aspect of the videos. During the replay of a game session's demo file, the character animations sometimes exhibit jerky motions, and this effect is exacerbated during a character's close up shot. Because we did not have access *UT3*'s underlying game engine source code, it was not possible to make an attempt to remedy this shortcoming.

Afterthought identifies complex interactions between players in 3D games and generates videos showcasing them. A preliminary evaluation was performed to determine the effectiveness of Afterthought's dynamic camerawork during a replay. The data gathered supported the claims that the custom camerawork provided for more humorous and more dramatic cinematics in comparison to the standard third-person view. Qualitative text responses from subjects included both positive remarks and constructive criticism indicating where improvements need to be made.

## References

Arijon, D. 1991. *Grammar of the Film Langauge*. Silman-James Press.

AviSnth software [On-Line]. Available via WWW. URL:http://avisynth.org/mediawiki/MainPage.

Cheong, Y.; Jhala, A.; Bae, B.; and Young, R. M. 2008. Automatically generating summarizations from game logs. In *AIIDE-08*, 167–172.

Friedman, D.; Feldman, Y.; Shamir, A.; and Dagan, T. 2004. Automatic creation of movie summaries in interactive virtual environements. In *VR '04*, 191–199.

Halper, N., and Masuch, M. 2003. Action summary for computer games: extracting action for spectator modes and summaries. In *Proc. of the Conference on the Application and Development of Computer Games*, 124–132.

Virtual Dub [On-Line]. Available via WWW. URL: http://www.virtualdub.org/.

YouTube Uploader [On-Line]. Available via WWW. URL: http://www.dvdvideosoft.com/guides/dvd/upload-video-to-YouTube.htm.