Automated Creation of Movie Summaries in Interactive Virtual Environments

Doron Friedman
Department of Computer Science
University College of London
d.friedman@cs.ucl.ac.uk

Yishai A. Feldman
Efi Arazi School of Computer Science
The Interdisciplinary Center, Herzliya
yishai@idc.ac.il

Ariel Shamir
Efi Arazi School of Computer Science
The Interdisciplinary Center, Herzliya
arik@idc.ac.il

Tsvi Dagan
Department of Computer Science
The Academic College of Tel Aviv Yaffo

Abstract

Virtual environments and artificial worlds are becoming multi-user, complex, and long lasting. Someone who was away from the environment for a while may wish to be informed of interesting events that happened during her absence without watching hours or even days of interaction. A movie is the natural medium for such a summary. At the end of a long interaction, participants may wish for such a movie as a keepsake. However, creating a movie summary of a given set of events is delicate, complicated, and time-consuming. In this paper we discuss some of the issues involved in creating a tool that can create such movie summaries automatically. We describe the stages in transforming a chronicle of events into a movie and present an implementation of such a system. We show results of transforming a log from a life-simulation game to a movie script.

1. Introduction

More and more people are spending longer times playing computer games and visiting virtual environments. As technology advances, these virtual worlds are becoming more complicated and more realistic. Spending time in them is much like travelling to fantastic destinations. As in real-life journeys, people would like to preserve some of their own personal memories and experiences from these places. What better souvenir is there than a short movie displaying the highlights of their journey? Similarly, online multi-user computer games can be played in persistent virtual worlds that may exist for years, and may be shared by thousands of people. When a player logs on to the game after a few days, she may want to see what she missed in the form of a short movie summary.

There is clearly a need for various types of summaries and replays of interactions in virtual environments. Intelligent summaries may also be useful for training simulations, educational applications, and in fact almost any type of virtual environment used over a long period of time. One would not want, or sometimes cannot, view a replay of the whole interaction session. A short movie summarizing the highlights and important events is sufficient and many times is even preferable. Furthermore, due to the large amount of people involved in virtual environments and games, and the need to add their personal points of view of the events, it is unrealistic to expect that such summaries will be produced manually. There is a need for *automatic* tools that can extract the interesting highlights out of long interaction sessions, manipulate them, and present them as movie clips that can be easily understood and are fun to watch.

We have identified two main stages in the automatic creation of high-quality movie summaries from complex environments. The first is the *extraction* stage, which involves decisions of what we want to show in the summary. The second is the *production* stage and involves decisions on how to display it in the best possible way. In fact, the extraction and production stages are not totally independent, since how you show some scene is closely related to what you choose to show and vice-versa. Nevertheless, it is useful to address them separately. A schematic overview of the whole process can be seen in Figure 1. The log of all events from a virtual world is the input to the *narrative extractor* (NE), which analyzes the input session log and tags the interesting parts. This information is sent into the *movie constructor* (MC), which decides how to concatenate the tagged parts into a coherent movie. The movie constructor can manipulate important events, remove unimportant events and (as will be seen later), can also add some artificially-created events. The output of the movie constructor is still a series of events augmented with some information, which we call the *script*.

A script is the usual medium that describes what is happening in a movie before it is shot. Our script is used as the input to the *animation generator* (AG), which creates the actual movie. It is easy to see that simply concatenating scene fragments together will not create an appealing movie. Moreover, there is no reason to replay the events from the same angle the user saw it originally. For example, in a multi-user environment, a user may wish to review how the scene looked from another participant's point of view, or from an overview that covers a wide range. The animation generator includes a knowledge-based cinematography system, which makes cinematic decisions about camera behavior.

In this paper we will concentrate on the first two parts of the process: the extraction part, and the movie construction part. More details on the animation generator and the production part can be found in [1]. A pre-requisite of the extraction stage is the availability of some form of a chronicle, or log. This log describes *all* events that took place during an interaction over some period of time in a virtual environment. There are many issues involved in defining, gathering, and ordering such chronicles of highly distributed scenes such as virtual environments. We defined a simple format that is general enough to apply to most environments and games, and manually created artificial logs of events inspired by a life-simulation game such as The Sims (`http://thesims.ea.com`). In such a game, simulated characters are engaged in everyday activities. Some of the actions of the characters are decided autonomously based on a model of need-satisfaction, and some are controlled by the player's commands. We created a small environment in VRML to model such a game environment with different characters and objects. (see Figure 2).

The main task of the extraction stage is to distinguish between interesting and non-interesting events. This requires a computational theory of what is interesting. In its most general form, this problem is obviously beyond the capabilities of the state-of-the-art of artificial intelligence. We rely on Schank and Abelson's theory of scripts, developed in the context of automatic story understanding [2]. They define a script as a set of routine actions. One of the examples they provide is eating in a restaurant. When you are told about a person who went to a restaurant, you can implicitly assume that a large number of things have happened: the person waited for the hostess, sat at a table, read the menu, ordered, received his order, ate, paid, left a tip, and then left the restaurant. If you read about a person going to a restaurant in a book, you will not be provided with all this information, or else you will find the book very boring. Instead, you will be provided information about the *unusual* things that may have happened in this restaurant visit, such as that the food was not served, or that the person decided not to leave a tip.

Schank and Abelson suggested the mechanism of scripts as a tool for story understanding. We suggest using scripts as an aid for deciding what is interesting in a log of events. We will typically be interested in scripts if they do not take place as they are supposed to; that is, we are especially interested in near-missed scripts. We describe an algorithm to find such scripts in the log and tag them as interesting. Other specific pre-defined actions or characters are also used to tag events as interesting in creating the tagged log.

Simply removing all untagged events from the log can cause serious problems both in terms of the consistency of events that occurred and in terms of the story line and movie creation. To overcome these problems, we introduce mechanisms that can modify events (create ease-in and ease-out of actions) and even insert new artificial events.

2. Related Work

Surprisingly, virtual environment summarization has not been addressed by the research community. Some work based on a similar motivation was done by Halper and Masuch [3]. They allow action summarization as well as live viewing of action in multi-player games. Their definition of what is interesting is very different from ours; they look at the derivative of a function that samples some story related parameters over the duration of the game. They admit that this method is tuned for action games, and may be limited for other types of virtual environments, such as strategy games, where actions can have long-term effects. Also, they did not address the issue of movie construction after the interesting parts were detected.

Substantial research is currently dedicated to automatic video summarization, but most of the research is focused on the very difficult problems of image analysis and understanding [4, 5]. In our domain the input is on the much more convenient level of abstraction: objects and actions, rather than pixels. If a system is able to extract this kind of information from video sequences, it could then benefit from our approach for creating automated summaries; this would serve a very wide range of applications.

Our research can be thought of as a new incarnation of research in automated story understanding. This is a problem that was investigated in the early days of artificial intelligence. As it proved to be a very difficult problem, it was either abandoned, or incorporated into research in natural language understanding [6]. Our context serves to separate the challenge of natural language understanding from the challenge of story understanding, since the input to our system is a formal account of events, rather than a natural-language description. Our research is thus mostly inspired by relatively early work in artificial intelligence, such as Schank and Abelson [2].

Some of the more recent research dedicated to automated text understanding does try to address the problem of what makes a piece of information interesting. For example, Byrne and Hunter [7] try to extract useful information from news reports by trying to detect violations of expectations. Their system accepts structured news reports as inputs, translates each report to a logical literal, identifies the story of which the report is a part, looks for inconsistencies between the report, the background knowledge, and a set of expectations, classifies and evaluates these inconsistencies, and outputs news reports of interest to the user together with associated explanations of why they are interesting.

Movie summaries play a significant role in many video games, especially sports games. However, these summaries usually do not go beyond simply showing a single event with an obvious meaning, such as a goal in a soccer game, or the player being killed in an action game. There is no attempt at story understanding, and no attempt at collecting various scene fragments into one coherent movie summary. An interesting recent trend is the new genre of filmmaking called Machinima (`http://www.machinima.com`): hobbyists producing animated movies using video game engines. The popularity of this trend further indicates the need for easy-to-use tools for the creation of movie summaries.

3. Narrative Extraction

The input to the narrative extraction stage includes three components:

1. the interaction session log of actions and events that took place during a scene;
2. a description of the virtual space including all characters and objects; and
3. predefined information for filtering the interesting parts of the session.

We first illustrate how narrative extraction works using a simple example. Our example will include one character called Joe, who carries out his morning activities and goes to work. For simplicity, our scene happens in one room (such as a large studio flat). The room includes a toilet, a shower, a garbage can, a table, and a chair. This is described in a textual format, including the positions and orientations of the objects. The log of this example scene is as follows:

```
00 00 Joe sit       Chair2
00 00 Joe wake-up
00 02 Joe move walk p1
02 04 Joe move walk p2
04 09 Joe use-bathroom Toilet
10 12 Joe move walk p3
12 13 Joe jump
13 16 Joe move walk p2
16 18 Joe move walk Shower_op
18 22 Joe take-a-shower Shower
24 26 Joe move walk p4
26 28 Joe move walk p5
28 30 Joe move walk Chair2_op
30 32 Joe look-at   Chair2_up
32 34 Joe sit       Chair2
34 36 Joe stand
36 38 Joe move walk p6
38 40 Joe move walk p7
40 42 Joe move walk Food_op
42 44 Joe prepare-breakfast Food
44 46 Joe move walk p8
46 48 Joe move walk p9
48 50 Joe move walk Chair2_op
50 52 Joe look-at   Chair2_up
52 54 Joe put Food  Table
54 56 Joe sit       Chair2
56 64 Joe eat       Food
64 66 Joe stand
66 68 Joe take      Food
68 72 Joe move walk p6
72 74 Joe move walk p10
74 76 Joe move walk p11
76 78 Joe move walk p12
78 80 Joe move walk GarbageCan_up
80 82 Joe drop-garbage Food Can Bottom
82 84 Joe look-at   p13
84 86 Joe speak     "Bye, honey!"
86 88 Joe move walk p14
88 92 Joe go-work   Car
```

Each line of the log describes an action, beginning with its start and end times. Thus, the original scene takes a minute and a half (note that this is only an illustration; in real life it should probably take more than two seconds to prepare breakfast). `p1` to `p14` are specific locations in the room, which describe Joe's walking path. It is clear from this log file that Joe performs a morning routine including going to the toilet, showering, and having breakfast. It is also clear that rather than performing these acts in the most efficient way, Joe occasionally wanders around the room or sits down pointlessly.

While this scene in itself is not very exciting, it can be seen as a typical fragment of a life-simulation game. This log is converted into a data structure in memory, called the *intermediate representation* (IR). A simple way to think about the IR is as a list of actions placed on a time-line. Each action includes several arguments such as actor, positions, angles, and target objects. Using the IR we can specify different filtering criteria to create different summaries based on the same scene log. There are four types of filtering criteria:

1. Specific events that should be tagged as interesting.
2. Specific characters whose actions should be tagged as interesting.
3. Composite boolean combinations of the above.
4. Definition of routines that should be recognized in the log.

For example, we can specify that we are interested in a specific action, such as eating. This will result in all eating actions in the IR being tagged. If we specify that we are interested in a specific actor, all the actions performed by that actor will be tagged. Composite rules extend the tagging by allowing boolean combinations of both actor and action such as: "I'd like to see all the cases where Joe is having a shower," or "I'd like to see all eating actions except for the ones that involve Jill." The tagging procedure for such criteria is straightforward: the IR is scanned, and the actions matching one of the criteria are tagged.

The fourth filtering criterion is more complicated. We allow the definition of sets of actions, which we call *routines*. Routines are an abstraction of actions. In some occasions, such routines indicate that the story can be considered interesting, such as a set of actions that imply that an actor is being robbed. However, often a story is interesting when in fact there is a deviation from a regular routine. For example, we can define a morning routine to include the following actions: taking a shower, going to the toilet, and having breakfast. If Joe performs the same routine every morning, it would not be considered interesting (except, possibly, for the first time). However, if Joe were to go through the morning routine without taking a shower, that could have (unpleasant) implications on the story development. This would make the whole morning session worth watching.

Routines are more than just a set of actions; they are a set of actions with causal dependencies. While we do not have information about causality in the session log, we do have temporal information. We can plausibly assume that a routine is a set of actions that take place through a scene, with some temporal restrictions. We assume each routine has a first action, a last action, and several intermediate actions. The intermediate actions have partial dependencies, resulting in a directed a-cyclic graph (DAG). The textual description of the morning routine is as follows:

```
Routine: wake-up ~ go-work
Time-limit: 1200
Actions:
  prepare-breakfast, eat, drop-garbage,
  use-bathroom, take-a-shower
Dependencies:
  prepare-breakfast < eat
  eat < drop-garbage
Miss: 2
```

This routine assumes the same actor carries out all actions. The `routine` section constrains the routine to start with the actor waking up and end with the actor going to work. The time-limit requires that the whole routine will take no longer than twenty minutes to perform. The routine includes five internal actions, but they do not need to appear in a specific sequential order. Instead, the dependencies section describes any partial order requirements. In this case, eating can only happen after preparing breakfast, and dropping the garbage can only happen after eating. All actions in the routine are implicitly dependent on the frame start-action, and the frame end-action is dependent on all other actions. The `miss` argument will be explained below. The resulting DAG for this routine is shown in Figure 3.

Searching for a routine in the IR representing the session log is done using topological sorting. Assume $G = \{V, E\}$ is the DAG representing the routine. V is the list of nodes in the DAG, each one representing an action. E is the list of directed edges in the DAG, each one representing a dependency relation $e = \langle m, n \rangle$ where $m, n \in V$. The dependents of $m \in V$ are defined as $\text{dep}(m) = \{n \in V | \langle m, n \rangle \in E\}$.

The search for routines is performed by traversing the list of actions according to their order. Whenever a first action of a routine is found, this triggers a search to match this routine. This means that several searches for the same routine can be carried out in parallel if the first action is encountered more than once. The search for routine match ends either when a match is found or when the time limit of the routine is reached. We initialize $d(n)$ for each node $n \in V$ to be the in-degree of n (that is, the number of edges entering n). We define the zero-set $Z \subset V$ as all nodes with zero degree. Note that initially only the start-action has a zero degree, and therefore Z includes only the start-action. The search for a specific routine match is as follows:

```
begin loop:
extract next action $m$ from log
if $m$ is the end-action or time limit is reached then
    exit loop
if $m \in Z$ then
    remove $m$ from Z
    set $d(m) \leftarrow -1$
    for each $k \in \mathrm{dep}(m)$
        $d(k) \leftarrow d(k) - 1$
        if $d(k) = 0$
            insert k to Z
end loop
check for a match or near miss
```

If we reached the end-action in the routine, we can search for a match. A routine match occurs if for all $n \in V, d(n) < 0$. If there is no match, we may still be interested in near misses. Therefore, when the routine match search ends we define: $M = \{m \in V | d(m) \geq 0\}$. We further define $\delta = \sum_{m \in M}(d(m) + 1)$, which will give us an indication on the magnitude of the miss. Now, recall that each routine had its miss parameter. This parameter indicates if we are interested in an accurate match, and if not it specifies the miss tolerance we are interested in. We mark a routine as interesting only if one of the following cases apply:

1. $\delta = 0$, i.e. perfect match.
2. $0 < \delta \leq$ miss, i.e., a missed routine within the accepted tolerance.

We typically define the miss parameter to be 1 or 2 which means we are only looking at scenarios with a small deviation from a routine. We have found this to be a satisfactory criteria for interesting scenarios. Nevertheless, using this scheme means that longer routines have a smaller chance of creating a near-miss. We have also implemented a version that allows defining the miss parameter as a percentage of the sum of all in-degrees in the graph.

4. Movie Construction

So far we have shown how the actions in the log session are tagged as interesting. If we take this set of actions and simply concatenate them into a movie, the result will be very difficult to understand. We want to construct a coherent movie out of such a collection of isolated actions. In fact, early filmmakers in the beginning of the 20th century were faced by the same problem. They discovered that movie time does not need to be equal to story time. In our context, story time refers to the time actions took place in the virtual environment as given by the log session. Movie time refers to the time in which these actions are displayed in the resulting movie. Early filmmakers gradually discovered that they could show a scene over a period of time that is shorter than the duration in which it took place on the set. This was done by cutting out some of the action in the editing room. They discovered (or invented) the cinematic techniques that allow this manipulation of time to appear flawless for the viewers.

Manipulation of story time is of course not limited to shortening. We can define the story time as a set of discrete intervals S, and the movie time as a set of discrete intervals M. A manipulation of time can be any function that maps story time into presentation time. Time can be shortened or extended. The order of events (intervals) can be completely transformed. Also, some parts of the story can be omitted altogether. The presentation time may also include new intervals not matching any interval in the story time, as we may want to add events that did not actually happen in the original story. In fact, it became necessary to use such manipulation in our movie constructor as will be explained shortly. Our movie construction has a relatively modest goal: we want to include only the tagged actions from the log. Therefore, we only used two operations: omitting intervals and inserting intervals. More complex manipulations such as shortening, extending, or changing the order of intervals are left for future work.

4.1. Ease-in and Ease-out

If we simply concatenate the intervals with the tagged actions and display the result as a movie, we get what film makers call *jump cuts*. For example, if we shoot Joe walking from his bed to the bathroom and then stop the camera, and restart it from the same angle when he is already sitting at the table and eating, the result is a (correct) impression of a jump in time. Although this was deliberately introduced into the cinematic language by French film maker Goddard in the

1950s, it is still typically avoided by mainstream film makers [8].

Typically, when filmmakers contract the presentation time, they want the viewers to be aware that time had elapsed. One possible cinematic technique is to use transitions such as fade or wipe effects. In our context there are too many cases in which we want the system to connect two temporally-separated scene fragments, and we did not want to make excessive use of such transitions.

Another way to avoid the effect of a jump cut when contracting time, is to use significantly different camera angles before and after cuts. However, when time manipulation is involved, this might not be acceptable, specifically if the actors remain in the frame in a certain posture when cut, and then shown immediately in the next frame in a different posture. For example, if at one moment you see an actor sitting on the chair, and at the next fraction of second you see the same actor standing in another part of the room, this would still constitute the disturbing effect of a jump cut, even if the camera angle would be significantly different before and after the cut. This problem only arises when the time is manipulated.

Our solution to remove possible jump-cut problems was to add an ease-in phase if an actor is seen in two consecutive tagged actions. For such marked action, we search for the previous motion of the same actor, and tag it as well. If the previous action is long, we only want to show its last few seconds. We then interpolate the point on the walking path according to the time that we want to show. This allows us to find the correct position to begin, remove the previous segment and insert the new shorter segment. Likewise, given a tagged action, we can look for the next action of the same actor and tag it in a similar manner.

4.2. World and Story Consistency

Assume that the original session log included intervals $I_1, \ldots, I_n$, and the movie constructor decided to remove a section of intervals in the middle $I_j, \ldots, I_k$ such that $1 < j < k < n$. This means we need to concatenate the end of interval I_{j-1} with the beginning of interval I_{k+1}. The problem is that if we look at the state of the scene by the end of I_{j-1}, we might find that it is inconsistent with the state of the scene at the beginning of I_{k+1}. For example, by the end of I_{j-1}, we might have clothes scattered all around the floor in a room. During the intervals $I_j, \ldots, I_k$ the room was cleaned up. However, since we cut out these intervals, these actions will not be performed and the resulting movie would still include the clothes on the floor.

The reason for this abnormality is that once the tagged log is converted to a script and sent to the animation generator, the animation generator is completely ignorant of the original story and actions. Its only task is to choose camera positions and angles and create shots from the set of intervals of the script. This is reminiscent of the *continuity* problem in film making: a specific scene is typically composed of separate shots, which may have been filmed several times, and at different times. The editor needs to make sure that the edited version of the scene appears continuous in space and time to the viewer.

The solution to this is to process all removed intervals $I_j, \ldots, I_k$, and in fact perform all actions of characters and objects in the original session log, in the order of their appearance, but in zero duration. This is done by inserting a new zero-time event for each action in the intervals. Performing these zero-time actions will take care of world and story consistency, and their zero duration ensures that they will not appear in the movie itself. As a result actors and objects "teleport" to their new position, change their shape or state or posture instantly, etc. It is obvious that not all such actions need to be performed (think of a person pacing up and down a room). Nevertheless, since they all have zero duration, and our system works off-line to create the movie, we did not include any optimization mechanism.

Once all chosen intervals are tagged, the script is created by choosing, in serial order, all tagged actions from the log, and copying them. New beginning and ending times are assigned by accumulating the time for each action copied. Hence, the zero-time actions are copied to the script but do not contribute to the movie-time.

5. Examples and Results

Going back to our example of Joe's morning log and applying the morning routine extraction, tags all actions related to this routine. In this example, there is a perfect match and we decide to include it in the movie, although we would typically prefer to ignore such perfect-match routines. The resulting script is:

```
00 00 Joe sit
00 00 Joe wake-up
00 02 Joe move walk p1
02 04 Joe move walk p2
04 09 Joe use-bathroom Toilet
10 12 Joe move walk p3
12 12 Joe move walk p2
12 14 Joe move walk Shower_op
14 18 Joe take-a-shower
20 22 Joe move walk p4
22 22 Joe move walk p7
```

```
22 24 Joe move walk Food_op
24 26 Joe prepare-breakfast
26 28 Joe move walk p8
28 28 Joe move walk p9
28 30 Joe move walk Chair2_op
30 32 Joe look-at Chair2_up
32 34 Joe put Food Table
34 36 Joe sit Chair2
36 44 Joe eat Food
44 46 Joe stand
46 48 Joe take Food
48 50 Joe move walk np0
50 50 Joe move walk p12
50 52 Joe move walk GarbageCan_up
52 54 Joe drop-garbage
54 56 Joe look-at p13
56 58 Joe speak "Bye, honey!"
58 60 Joe move walk p14
60 64 Joe go-work Car
```

This differs from the original session log in several ways. Some actions were removed: Joe jumping (originally after 12 seconds), sitting near the table, and other actions that were not part of the morning routine. There are several zero-time actions and the overall scene time is reduced to one minute.

In the original session Joe wanders around with the food remainders before throwing them to the garbage, for some 10 seconds (68-78 in the original session log). The Narrative Extractor decided to remove this wandering about and replaced it by two walking actions: in the interval $[48, 50]$ Joe walks to np0, and in the interval $[50, 52]$ Joe is displayed walking to the garbage can. Hence, we introduced a cut in the original session, so we need the “teleport” action from np0 to p12 with zero duration to fill the gap. The action of walking to np0 did not exist in the original session. It was generated to avoid the jump cut after Joe got up from the chair. It is part of an action that took place in the original script; the original action took 4 seconds (68–72), and the NE decided to cut it into a 2 second interval. This is not a significant modification of the story line, but it illustrates the introduction of new actions to the script.

As can be seen, the log includes some high-level actions such as `wake-up` (standing after lying in bed) or `drop-garbage` (put an object in the garbage can). These are collections of more atomic actions that often appear together and can be recognized in the session log, and facilitate the specification and recognition of routines. When such actions are sent into the animation generation component, they need to be expanded back into lower-level components. Such high-level actions can be extracted from the virtual environment, and thus appear in the original session log, or can be introduced by the NE.

A second, more elaborate, example scene includes four actors. Jogger is a girl jogging around the house. Joe is performing his morning routine as in the previous example, and after about a minute and a half, he goes off to work. Just after Joe goes to work, a thief comes running into the house, grabs a fancy object (a wrapped gift), and runs away. A lady called Nana is casually hanging around the house all that time, until she suddenly spots the thief. This makes her scream and run out for help (see Figure 4). The whole scene now takes around two minutes.

Different versions of movie summaries can be generated out of this log depending on the filters specified. Consider a simple example when we tag the thief as an interesting actor, and watching TV as an interesting action. No routine is tagged, because we are not interested in a morning routine, unless it was a near miss. The resulting screenplay includes several zero-time actions. For brevity, we omitted those that have no effect on the resulting movie.

The resulting summary is half a minute long, instead of the original two-minute session. It includes parts of the morning session, since Nana is watching TV, which was tagged as an interesting action. Then it includes the whole burglary scene. Note that although we only focus on the thief, the movie will include all other important actions occurring at the same time. In this case it will also include Nana spotting the thief, shouting, and running away. Also, the jogger would appear in correct places at the right time. For example, in the original session the thief waits for the jogger to pass behind the house and out of sight before entering the house. This will also be displayed in the summary movie.

One deficiency of this summary is that watching the movie we will miss the point that the thief enters right after Joe leaves for work. This exposes a gap in our story understanding. A human who would be making this summary would understand this causality. Therefore, he would probably decide to show Joe leaving the house before showing the thief arrive in the scene. This is beyond our system’s current capabilities. In order to understand this information, we would need additional artificial intelligence capabilities, such as common-sense reasoning and the ability to infer causality relationships.

6. Conclusion and Future Work

Although we had to make several simplifying assumptions in the course of this work, we have managed to show that interesting and intelligent movie summaries can indeed be created automatically from virtual world session logs. Not surprisingly, we found that cinematographic knowledge is a crucial component of such an automatic system. Nevertheless, we also found that in many senses, generating virtual world summaries is different from editing movies. Specifically, there is a greater challenge to keep the world and storyline consistent.

We also found it extremely important to evaluate our methodologies by experimentation. Although we could easily theorize about the algorithms and transformations, we were often surprised by the results. Such algorithms should be evaluated empirically by receiving systematic feedback from domain experts (e.g., film makers) and from the target audience of a specific application (e.g., game players). We believe multi-user environments, and especially persistent, massive multiplayer, online games will especially benefit from our paradigm. If we think of these as virtual alternatives to our world, watching such summaries may be the essential equivalent of watching broadcasted newscasts in the real world.

The creation of more elaborate movie summaries from virtual world sessions is still a challenging and important problem. For example, we have illustrated only the very basics of temporal manipulation. Future work can explore how story time can be transformed into movie time in a more complex manner. Also, it will be interesting to see how we can automatically decide about scene boundaries, and how we can automatically intertwine scenes, for instance, as it is done in most soap operas.

We also believe that in the scope of a closed world domain (such as The Sims), we should be able to leverage the rich information provided by the application. These include the relationships between the characters, the characters' personalities, and their mood. It should also be possible to automatically infer some causality chains which will allow progress in two directions. First, we may be able to extend the narrative extraction component and its definition of what constitutes an interesting action. Second, a higher level of story understanding can result in higher-quality cinematic decisions.

Our implementation is limited in scope, specific, and domain dependent, but our approach can be generalized to many types of virtual environments. Our research highlights the need for a higher level of abstraction in representing virtual scenes. The objects, actors, events, and actions in each application are different. However, if they are abstracted, it should be straightforward to define routines based on these entities, and to generate a log of the events and actions.

While our case study is based on the specific-domain of life-simulation computer games, we believe the results are also of high relevance to immersive virtual reality research. Virtual reality systems are typically expensive and complex, and organizations spend significant resources on their construction. Thus, the sessions are very often video recorded for later analysis. For many applications, automated summaries can be useful, either instead of, or in addition to, these video recordings.

Finally, many researchers and artists are investigating the possibilities for non-linear narrative and interactive digital storytelling. Our research can be thought of as a complementary approach. We recognize that complex environments can generate multiple narrative threads. Our approach can thus be seen as the reverse engineering of narrative, rather than the authoring of it.

All examples in this paper can be viewed at `http://www.cs.ucl.ac.uk/staff/D.Friedman/sims`.

References

[1] D. Friedman and Y. Feldman, "Knowledge-based formalization of cinematic expression and its application to animation," in *Proc. Eurographics 2002*, Saarbrücken, Germany, 2002, pp. 163–168.

[2] R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals, and Understanding.* Erlbaum, 1977.

[3] N. Halper and M. Masuch, "Action summary for computer games: Extracting action for spectator modes and summaries," in *Proc. 2nd Int'l Conf. Application and Development of Computer Games*, L. W. Sing, W. H. Man, and W. Wai, Eds. Division of Computer Studies of the City University of Hong Kong, 2003, pp. 124–132.

[4] H. D. Wactlar, "New directions in video information extraction and summarization," in *10th DELOS Workshop, Santorini*, 1999, pp. 66–73.

[5] M. Brand, "The inverse Hollywood problem: From video to scripts and storyboards via causal analysis," in *Proc. 14th Nat'l Conf. Artificial Intelligence (AAAI-97)*, 1997, pp. 132–137.

[6] E. T. Mueller, "Prospects for in-depth story understanding by computer," 1999, manuscript.

[7] E. Byrne and A. Hunter, "Man bites dog: Looking for interesting inconsistencies in structured news reports," *Data and Knowledge Engineering*, 2003.

[8] T. Yaron, *Editing Movies.* Israeli Ministry of Culture, 1995, in Hebrew.

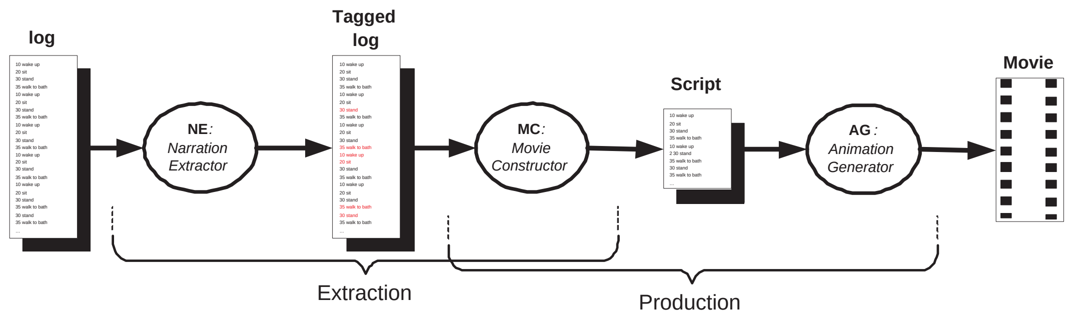

Figure 1. An overview of the stages in the system for automatic creation of movie summaries.

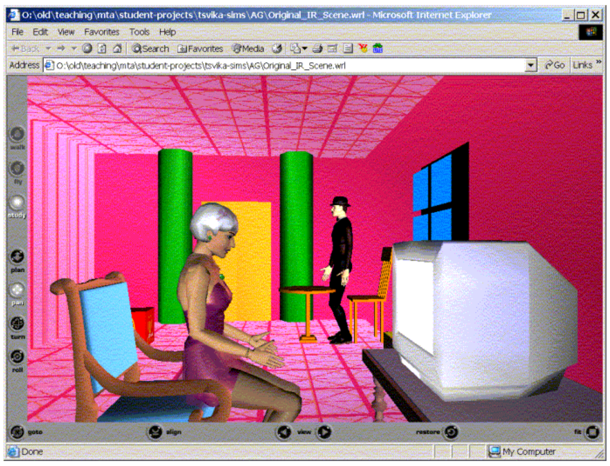

Figure 2. The VRML environment and characters used for modelling of a life-simulation game.

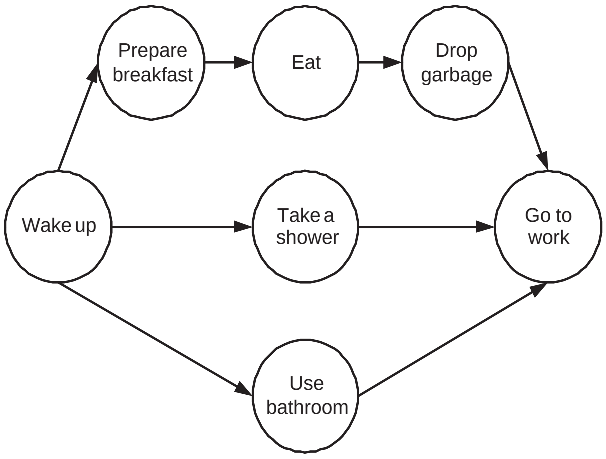

Figure 3. An example of a DAG representing a routine.

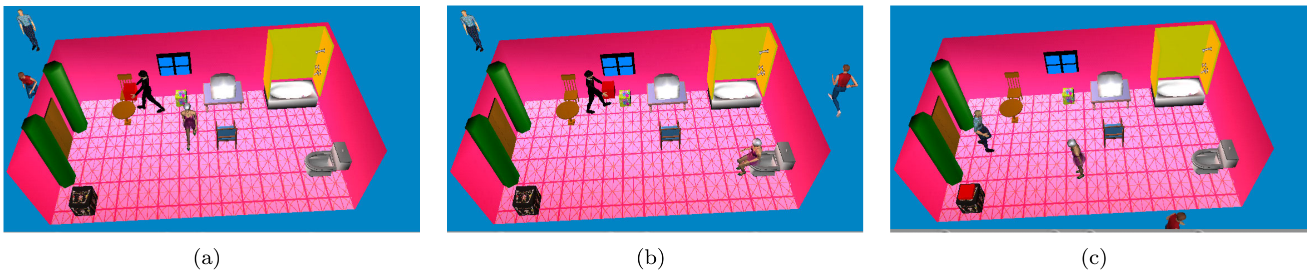

Figure 4. (a) Start of scene: girl is jogging; thief is lurking outside; inside the house morning routine is performed. (b) Middle of scene: girl is still jogging; inside the house morning routine is continued. (c) Towards the end of scene: Joe has gone off to work; the thief is entering the house; Nana is scared and runs outside; the girl is still jogging (see bottom of frame).