Automatic Camera Control Based On Avatar Behavior in Virtual Environment

Moon-Ryul Jung
Department of Computer Science
College of Information Science
Soongsil University, Korea

Abstract

This paper presents a method of controlling camera to present virtual space to participating users meaningfully. The users interact with each other by means of dialogue and behavior. Users behave through their avatars. So our problem comes down to controling the camera to capture the avatars effectively depending on how they interact with each other. The problem is solved by specifying camera control rules based on cinematography developed by film producers. A formal language is designed to encode cinematography rules for virtual environments where people can participate in the story and can influence its flow. The rule has been used in a 3D chatting system we have developed.

1. Introduction

In immersive virtual environment systems, the scene is typically captured from the viewpoint that is attached to the user's avatar. A user's avatar is the agent within space that represents the user outside of space. The avatar behaves in behalf of the user either by directly following the user's behavior, i.e. head movement and hand movement, or by interpreting the user's gestures, i.e. hand gesture or mouse movement. In virtual environment systems that support the sense of immersion in space, e.g. by means of the head mount display, the avatar's viewpoint is typically controlled by the head movement of the user. In other words, the space is always seen from the first person viewpoint, because it is required to support the sense of depth that is due to motion parallax.

Even if it is available, the immersive first person perspective alone is not suitable for virtual environments where users are supposed to respond to events, e.g. the appearance of another user in the environment. To respond to events, the users should be able to recognize their occurrences even when the events occur outside of the present field of view. One way to indicate occurrences of events outside of the field of view is to associate 3D sounds with events. When the user hears a 3D sound, the user can recognize where that sound has originated, and thereby can change the viewpoint to observe the event. Hence at least 3D sound capability seems to be required if we want to use the immersive first person perspective for virtual envirionments where users interact with the environment and with each other. But at the moment, 3D sound is too expensive to support at least for ordinary PC applications. There may be many virtual envirionment applications, e.g. 3D chatting, where the immersive first person viewpoint is not cost effective. Therefore, the immersive first person viewpoint is not yet the method of choice for camera control for virtual environments.

When the sense of immersion is not supported, the scene is seen through the computer screen without any 3D output devices. In that case, we often use viewpoints other than the first person viewpoint, because they are often more effective in communicating to users what is going on in the environment. This paper describes a method to control the viewpoint of the camera for virtual environment systems where the immersive first person viewpoint is not used.

The first person viewpoint is attached to the head of the user's avatar. So, in the case of the first person viewpoint, it is automatically determined by the posture of the avatar, which is in turn controlled by the avatar's body movement. Hence the user does not need to control the avatar's viewpoint explicitly. But to use other viewpoints, they should be controlled in addition to the avatar's body movement. However, it is hard for the user to control the viewpoint of the camera as well as the avatar's body movement at the same time. For example, in 3D chatting systems where users talk to each other through their avatars in the same space, users are busy with typing sentences and indicating behaviors of their avatars. It would be too much if they are required to control the viewpoint as well. In such systems, the camera should be controlled automatically so that users may concentrate on their main task, i.e. interaction with other users. Automatic camera control comes down to automatically computing the position and orientation of the camera so that the user may understand better what is going on in the scene and may feel less bored at the scene. To do so, we need camera placement rules for determining appropriate camera positions and orientations so that the resulting scene may cover important states and events occurring in the environment. In this paper, we design those rules based on cinematography, a collection of heuristic camera placement rules developed by film producers. Cinematography contains camera manipulation rules for effective scene composition and flow of story. To use the rules for virtual environment applications, we need to encode them in a certain form.

To encode cinematography rules for virtual environment applications, three issues, that is, language, context-dependency and user-dependency should be addressed. First, cinematography manuals, e.g. Arijon[1] and Mascelli[4], provide heuristic camera placement rules for various situations, e.g. a situation of three persons talking. Arijon [1] uses the concept of line of interest for a given situation relative to which cameras are placed. The line of interest is the line connecting the heads of the two primary actors. The line of interest divides the plane into two sides. It is believed that the audience wants to observe the actors on the line of interest from a viewpoint in one side of the plane. He[3] tried to encode cinematography by means of a state transition automata. But this encoding has been done without analyzing the problem of automatic camera control sufficiently. The language used to encode cinematography rules is not clearly defined, and many important concepts and parameters are hidden in the C code. We need an intuitive but a sufficiently fine-grained language that provides a framework in which expert knowledge for camera placement can be encoded easily.

Second, traditionally cinematography rules are developed for films whose scenarios are predetermined. So a sequence of camera shots for a given film is also predetermined. David[2] suggested a method to plan a sequence of camera placements for a given sequence of scenes. But in virtual environments, the flow of events is not predetermined, so that a sequence of camera shots cannot be pre-planned and should be determined in a context-dependent manner. A given application needs a procedure that determines the camera placement for the current moment based on the current situation. This is what we call context-dependency. Third, in films, situations are taken from three kinds of viewpoints. One is the viewpoint of a participant of the story. The other is the quasi participant viewpoint which captures the scene from a viewpoint around a participant, e.g. over the shoulder of a participant. The last viewpoint is the neutral viewpoint or non-participant viewpoint which captures the whole situation in a neutral viewpoint. In virtual environment applications where the immersive first person viewpoint is not supported, the scene is taken basically in the same manner as in films. But because the audience is also an actor that participates in the scene, there are some limits in making the audience assume the roles of participants. For example, when two persons are talking, the user that speaks would like to see the reaction of the hearer, rather than his own face. The speaker already knows what his avatar is doing, because the speaker dictates the avatar's behavior. As long as the user speaks, it is not natural to make him assume the role of the hearer. In the same reason, the user whose is the hearer would like to see the speaker rather than his own behavior. This is what we call user-dependency. He[3] did not consider user-dependency in encoding cinematography rules. When encoding camera placement rules, we need to take it into account.

In this paper, camera placement rules are devised having in mind the three issues, i.e. language, context-dependency and user-dependency. The rules are designed for a 3D chatting system.

2. 3D Chatting System

In this paper, camera placement rules are designed for a 3D chatting system. So, the overall structure of the system is briefly described.

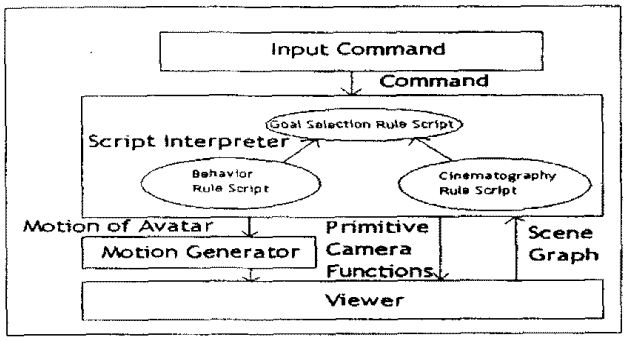

Figure 1: The overall flow of the 3D chatting system

The whole system, shown in figure 1, works in a typical event-driven manner. Each user types in sentences and action commands for his avatar. The system puts the sentences in dialogue bubbles as in cartoon and generates the behavior of the avatars as requested by the users. The behavior of avatar User is produced by calling a callback function exec_scipt(User) and thus executing the behavior script written for avatar User. The callback function is called when there are no other events pending. The function tells the script interpreter to execute the behavior script. Also, when there are no events pending, the system calls a callback function exec_script(camera), which tells the script interpreter to execute the script for camera. The camera script consists of goal selection rules and goal achievement rules. Goal selection rules are used to select a camera placement goal whose achievement would capture the current situation well. Goal achievement rules determine camera placement subgoals and primitive camera actions contributing to the achievement of the selected camera placement goal. Each time the camera script is executed, a cycle of executing the goal selection and achievement rules is performed. The cycle is finished by executing primitive camera actions.

3. Formal Specification of Cinematography

This paper tries to encode cinematography in a formal language. So, we define rigorously basic concepts and principles of cinematography necessary for formal encoding. Cinematography rules are reviewed based on Arijon [1], a comprehensive manual for camera placement rules. Arijon describes camera placement formulas for various situations, e.g. one person moving in the scene, one person entering the scene, two persons talking, three persons talking, and more than four persons talking. Mathematically speaking, a camera placement is defined in terms of two parameters, i.e. the position and the orientation of the camera. But the position and orientation parameters are too low level to be meaningfully controlled in camera placement rules. So, Arijon uses somewhat vague but more intuitive high level parameters to specify camera placements. In short, camera placement is specified in terms of the line of interest, camera angle, and camera distance.

3.1 The line of interest

The most important concept is the line of interest, because camera angle is specified relative to it. In Arijon[1], the line of interest is taken to be the line of sight in the case of a single actor. In the case two persons interacting, e.g. talking, the line of interest is taken to be the line connecting the heads of the two persons. The line of interest is basically the line that the camera, that is, the audience, would want to observe. Typically, some actors on the line of interest are given more camera attention than are other actors. It is usually determined by the roles of the actors. Also, the audience would sometimes want to observe the whole situation from a neutral point of view to establish the context of the current event. The line of interest is defined even when two persons do not look at each other. For example, when two persons are talking while looking at the same direction, the

line connecting the heads of the two persons is the line that the audience would like to observe.

3.2. Camera angles

In film, multiple cameras are used to support multiple camera placements when the transition from one placement to another is discontinous, that is, when the camera position or orientation is jumped to another one. In such cases, it is physically or temporally not feasible to use a single camera. So, cameras to be used to capture a scene are placed at predetermined angles.

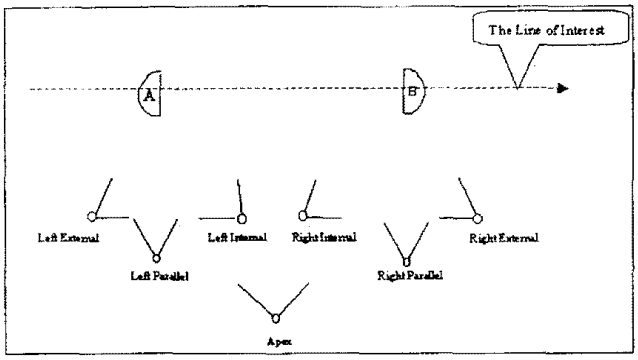

Figure 2: Seven camera angles

Arijon suggests that we posit seven kinds of camera angles relative to the line of interest between actor A and actor B, as shown in figure 2. The line of interest divides the plane into two sides. The line of interest has no direction, because it simply connects the two primary actors. But for the purpose of dividing the plane into two sides, the line of interest between A and B is viewed to have the direction from A to B. The two side are considered the left and right sides, respectively, from the viewpoint parallel to the line of interest. The cameras in figure 2 are all placed on the right side. There are also seven kinds of camera placements on the left side.

Let LOI be the unit directional vector corresponding to the line of interest lying between actors A and B. Camera Cam is placed at parallel angle with actor X (A or B) on the side Side (left or right) with respect to LOI (camera_placed_at(Cam, LOI, Side, parallel, X)) if the camera points to actor X from the side Side with its direction perpendicular to LOI. Camera Cam is placed at apex angle with A and B on the side Side with respect to LOI (camera_placed_at(Cam, LOI, Side, apex, A, B)) when the camera points to the middle of actors A and B from the side Side with its direction perpendicular to LOI. Camera Cam is placed at external reverse angle from A to B on the side Side with respect to LOI (camera_placed_at(Cam, LOI, Side, external, A, B)) when the camera points to actor B from behind A on the side Side with respect to LOI, so that actor B is covered fully and actor A is covered partially. Camera Cam is placed at internal reverse angle from A to B on the side Side with respect to LOI (camera_placed_at(Cam, LOI, Side, internal, A, B)) if the camera points to B from between A and B on the side Side with respect to LOI, so that only actor B is covered fully.

A given side is divided into two areas, one containing A and the other containing B. The two regions are called the left and right regions from the viewpoint of a camera positioned in the given side. Hence, as shown in figure 2, cameras placed on the side of A are called left cameras, and camera placed on the side of B are called right cameras. So, when camera_placed_at(Cam, LOI, Side, parallel, A) holds, the camera Cam is called the LeftParallel. When camera_placed_at(Cam, LOI, Side, parallel, B), the camera Cam is called the RightParallel. When camera_placed_at(Cam, LOI, Side, external, A, B) holds, the camera Cam is called the LeftExternal. When camera_placed_at(Cam, LOI, Side, external, B, A), the camera Cam is called the RightExternal. Similary for the LeftInternal and the RightInternal.

3.3. Use of multiple cameras in virtual environments

In virtual envirionments, there are no physical or geometric constraints on camera movement. So, any kind of camera placement can be immediately achieved. But if that happens, users would be confused because they are accustomed to films where camera movement is constrained. So, even for virtual environment, we want to follow ordinary constraints on camera movement and use multiple cameras for multiple placements. It is advantageous in two

aspects. First, the process of encoding camera placement rules becomes easier because there is one-to-one correspondence between cinematography manuals and encoded formulas. Second, maintaining multiple cameras is good for real-time camera control. If we use a single camera, then the position and orientation of the camera should be computed from scratch every time the camera placement transition occurs. But if we use multiple cameras, the new placement for a given camera may be obtained by modifying the previous placement incrementally.

For a given line of interest, some or all of these seven placements can be used. When some camera placements are to be used, they are initialized by assigning the default position and orientation to each placement. To do so, we use camera actions of the form

initialize_camera(Camera, LOI, Side, CameraAngle, A, B).

For example,
initialize_camera(Cam, LOI, right, external, A, B)
initializes camera Cam to a typical external angle from A to B on the right side with respect to LOI. Initialized camera are effective when they are activated. So we use camera actions

activate(Cam) and deactivate(Cam)

to activate and deactivate camera Cam.

3.4. Camera actions

Camera actions are classified into primitive actions and complex actions. Primitive actions simply change the position and orientation of a given camera by a certain increment each they are executed. We use the following primitive camera actions.

move(Cam, forward, D): move camera Cam along its forward axis Dir by distance D. Similarly for move(Cam, up, D) and move(Cam, left, D).

pan(Cam, Ang): pan (rotate about the vertical axis) camera Cam by angle Ang.
tilt(Cam, Ang): tilt (rotate about the horizontal axis) camera Cam by angle Ang.
roll(Cam, Ang): roll (rotate about the forward axis) camera Cam by angle Ang.

Complex actions are names of procedures that achieve camera placement goals, e.g. camera_placed_at(LeftExternal, LOI, Side, external, midium, A, B). Complex camera actions are expressed in the form of

place_camera(Cam, LOI, Side, CameraAngle, CameraDistance, A, B).

For example, place_camera(LeftExternal, LOI, right, external, midium, A, B) is the action that places camera LeftExternal at external angle from A to B with mididum distance on the right side with respect to LOI. Complex actions are implemented by a combination of primitive actions.
l

4. Encoding cinematography rules

Every client system in a 3D chatting system has its own user, which has logged in the client system. The camera script is shared by every client system. But the user of each client system has a different role in a chatting situation. For example, one user would be a speaker, and another user would be a hearer. The camera placement rules for the speaker are distinct from those for the hearer. So, even though the same camera script is used, different part of it would be executed depending on the role of the user. We show some of the goal selection and achievement rules used in our 3D chatting system.

Both goal selection and achievement rules are specified in the form of production systems. A production system is a sequence of conditional actions, which are written as

C1 => A1;
C2 => A2;
.....
Cn => An.

Every time a production system is activated, the conditions are tested in sequence, and the first action whose condition is true is executed.

4.1 Goal selection rules

We describe an example camera script in an informal

language.

```
script(camera) = {
  Let Group be the set of other avatars:
  Let User be the avatar of the user that uses the
      client system:

  (1) User appeared in the conversation room less
      than OverviewDur frames ago
     =>camera_for_overview(User, Group,
                           OverviewDur);

  (2) User is speaking to avatar B
     => camera_for_speaking(User, B, TalkCycle);

  (3) User is hearing avatar B
     => camera_for_hearing(User, B, TalkCycle);
}
```

The above camera script describe three situations and the camera placement goals for them. Rule (1) states the camera placement goal (camera_for_overview(User, Group, OverviewDur)) to be achieved and maintained when the user first appears in the conversation room. OverviewDur is the time interval during which the user is supposed to overview the new situation he has entered. The time interval is counted as the number of frames, which is incremented every time the camera script is executed. Rule (2) states the camera placement goal for the situation in which the user is speaking to avatar B. Similarly for rule (3). The above rules describe what to do when, and so are called goal selection rules.

4.2 Goal achievement rules

Here we define the camera placement goals mentioned in the goal selection rule above. The goal camera_for_overiview() is defined as follows.

```
camera_for_overview(User, Group, OverviewDur) = {
   Increment frame_count modulo OverViewDur:
(1) temporal_phase(frame_count, OverviewDur,
       0, 1/2) &&
    camera_placed_at(LeftExternal,LOI,
                     right, external,longshot,
                     User, Group) => nil:

(2) temporal_phase(duration_count, OverviewDur,
       0, 1/2) =>
     place_camera_at(LeftExternal,LOI,right,
                     external,longshot,
                     User, Group);

(3) temporal_phase(frame_count, OverviewDur,
       1/2, 1) &&
     camera_placed_at(LeftExternal,LOI,right,
                      external,fullshot,
                      User, Group) => nil:

(4) temporal_phase(frame_count,
                   OverviewDur, 1/2, 1)
   =>place_camera_at(LeftExternal,LOI,right,
                     external,fullshot,
                     User, Group);
}
```

The above rule says that camera_for_overview() is achieved by doing different actions depending on where the current time frame_count falls within OverviewDur. Roughly speaking, camera_for_overview() is achieved by taking a **long shot** at external reverse angle from User to B initially and then by taking a **full shot** at external reverse angle from User to B later. These two shots are intended to show the whole situation of the conversation room to the user which has entered the room. Incrementing frame_count modulo OverViewDur gets the remainder of dividing frame_count by OverViewDur. Predicate temporal_phase(frame_count, OverviewDur, 0, 1/2) means that frame_count falls within the first half of OverviewDur. In other words, it means that the current frame falls within the first half of cycle period OverViewDur. Rule (1) says that no camera action is performed if camera_placed_at(LeftExternal, LOI, right, external, longshot, User, Group) is true in the first half of the period. Rule (2) says that if camera_placed_at(LeftExternal, LOI, right, external, longshot, User, Group) is not true during the first half of the period, camera action place_camera_at(LeftExternal, LOI, right, external, fullshot, User, Group) is executed. Rules (3) and (4) state the similar things with respect to the second half of the period. Place_camera() is a complex camera action which is achieved by a combination of primitive camera actions.

Rule (1) is executed every frame, because the goal condition of rule (1)

camera_placed_at(LeftExternal,LOI,right,external,
longshot, User, Group)

may be false in the current frame even though it was true in the previous frames. In that case, rule (2) is fired, and the camera action to make the condition true is executed. All the camera actions mentioned in the goal achievement rule for camera goal

camera_for_overview(User, Group, OverviewDur)

are activated when the corresponding triggering conditions are true. Each action is supposed to contribute to the achievement or maintenance of the camera goal

camera_for_overview(User, Group, OverviewDur).

The goal is achieved when one of the conditions in the rule whose actions are nil.

This style of program is called "teleo-reactive" program [5], because what to do for achieving a given goal is determined in reaction to the current situation, and whatever the program does, it is to achieve the given goal, that is, one of the conditions whose actions are nil. Teleo-reactive programming is appropriate for camera placement goals, because camera actions are sensitive to the behaviors of avatars. However, goal selection rules are not teleo-reactive programs, but simply reactive, though they are written in the same production system. It is because there are no particular conditions in a goal selection rule toward which all actions in it are striving to achieve.

Let us see how the second camera goal in the goal selection rule

camera_for_speaking(User, B, TalkCycle)

is achieved. Parameter TalkCycle is the cycle period in which the camera captures various aspects of the situation in sequence. When the cycle period is expired, the camera returns to the initial point in the sequence and repeats the camera actions assigned to the cycle period. The second camera goal is defined as follows.

```
camera_for_speaking(User, B, TalkCycle) = {

Increment frame_count modulo TalkCyclel

temporal_phase( frame_count, TalkCycle, 0, 4/6 )
&& camera_placed_at(LeftExternal,LOI,right,
                    external,closeup,
                    User, B) => nil:

temporal_phase( frame_count, TalkCycle, 0, 4/6 )
  => place_camera(LeftExternal,LOI, right,
                  external, closeup,
                  User, B):
 temporal_phase( frame_count, TalkCycle, 4/6, 5/6
)
  &&camera_plaed_at( LeftExternal, LOI, right,
                     external, closeup,
                     User, B) => nil:

temporal_phase( frame_count, TalkCycle, 4/6, 5/6 )
  => place_camera( LeftExternal, LOI, right,
                   external, closeup,
                   User, B):

temporal_phase( frame_count, TalkCycle, 5/6, 1)
  && camera_placed_at( ApexCamera, LOI, right,
                       apex, closeup,
                       User, B)  => nil:

 temporal_phase( frame_count, TalkCycle, 5/6, 1)
=>place_camera( ApexCamera, LOI, right,
                apex, closeup, User, B):
}
```

Let us see how the third camera goal in the goal selection rule

camera_for_hearing(User, B, TalkCycle)

is achieved when hearer User is hearing actor B speaking. The line of interest goes from User to speaker B. As in camera_for_speaking(), during the first 4/6 of TalkCycle, the camera is placed at reverse angle from the hearer User to the speaker B. to capture the speaker. During the rest of the time, the camera is placed at reverse angle from the speaker to the hearer, to capture the speaker, and then is placed at apex angle with the hearer and the speaker to establish the whole situation from a neutral point of view.

5. Conclusions

In this paper, we have encoded camera placement rules for 3D chatting in virtual environments based on cinematography rules developed by film producers. We have designed a formal language for the encoding; we carefully devised predicates for camera placement and camera actions needed to specify rules or procedures to achieve desired camera placements for various situations in which users particiapte. The rules are encoded by taking into account context-dependency and user-dependency of cinematography in virtual environment. In the present study, cinematography rules for the interaction of multiple people, e.g. three persons talking, are not yet encoded. It requires a way to compute the line of interest for such situations. This problem needs a further systematic inquiry into the nature of heuristic knowledge of traditional cinematography.

6. Reference

[1] D. Arijon. "Grammar of the Film Language" Communication Arts Books, Hastings House, Publishers, New York, 1976.

[2] D. Christianson, S. Anderson, Li-wei He, D. Salesin, D. Weld, and M. Cohen. "Declarative Camera Control for Automatic Cinematography", AAAI '96.

[3] Li-wei He, Michael Cohen, David Salesin, "The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing" SIGGRAPH '96

[4] Joseph V. Mascelli, "The Five C's of Cinematography", Cine/Grafic Publications, Hollywood, 1965.

[5] Nils J. Nilsson, Stanford University, "Teleo-Reactive Program for Agent Control", Journal of Artificial Intelligence Research (1994), 139-158