A Virtual Cinematography System for First Person Shooter Games

Hugh McCabe and James Kneafsey

Insitute of Technology Blanchardstown
Blanchardstown, Dublin 15
Ireland
hugh.mccabe@itb.ie; james.kneafsey@itb.ie

Abstract. We present a virtual cinematography system for 3D shooter games. This system replaces the strict first-person view associated with such games with a camera system that can switch between multiple camera modes. The switching is carried by the system in real time as game-play progresses. The choice of which camera modes to provide, and the decisions about how and when to employ them, are informed by cinematography theory and practice. We argue that applying principles from cinematography to camera-work in computer games, results in a more interesting and engaging experience for the player. Our system has been implemented as a modification of the Quake II game engine.

Keywords: Cinematography, Games, First Person Shooter Games, Algorithms, Graphics, Programming.

1. Introduction

Computer games rely on the notion of a virtual camera that provides the player with a view of the action that is taking place within the game. Since the player is directly participating in, and to some extent controlling, the action, it is necessary to employ this camera in a manner that assists, rather than impedes, the activity of the player. Consequently the camera is typically used in one of a number of well-defined ways, each of which is designed to facilitate effective game-play. The two commonest approaches are to employ a third-person over the shoulder camera [5][6] or to fix the camera's view directly to that of the player. The latter approach is that taken by the well-known First Person Shooter (FPS) genre of games. With some notable exceptions (for example [19]) game developers have not found it worthwhile to explore the use of camera-work that strays far outside these conventions. It is also uncommon for the camera-work in computer games to be informed by, or directly influenced by, how real-world cameras are employed in film.

Virtual cameras however share many of the characteristics of their real-world counterparts, and hence virtually anything that can be done with a real-world camera can also be done with a virtual one. The craft of *cinematography* [2][15] outlines

principles and techniques pertaining to the effective use of cameras to film live action. The correct application of these principles and techniques produces filmed content that is more engaging, compelling and absorbing for the viewer. There are therefore clear advantages to successfully applying cinematography to camera-work in computer games, but the main barrier to doing so is the issue of *interactivity*. On a film set the action is carefully scripted and therefore the camera operators know exactly what is going to happen, where it is going to happen, and when. In an interactive game however, the software controlling the camera knows none of these things. When this is combined with the constraint of having to film the action in a way that facilitates the player's game-play, it's not surprising that the only area where film-like cinematography *has* made inroads is in the non-interactive cut-scenes inserted between segments of interactive game-play.

In this paper we explore the possibility of enhancing the camera-work in the interactive sections of shooter games by replacing the standard first-person camera with a *virtual cinematography* system. This system frees the camera from its fixed position at the avatar's viewpoint and instead employs a number of possible camera modes. As play progresses, the system monitors events occurring within the game-play, and chooses camera modes accordingly. The camera modes, and the rules which govern their employment, are informed by both cinematographic principles, and by practical considerations pertaining to the facilitation of the player's game-play. The result is a camera system that is capable of interactively cutting between different views in real time as game-play progresses. We maintain that such an approach can result in a more dynamic and engaging experience. We present both the design and implementation of the system and describe the results of testing it on a group of players.

The rest of our paper is structured as follows. Section 2 describes related work by other authors who have incorporated cinematographic principles to computer games. Section 3 gives a brief overview of some important concepts from cinematography with particular emphasis on those that have influenced our work. In section 4 we describe our virtual cinematography system in detail. Section 5 gives the results of testing the system on a group of players and finally section 6 presents our conclusions and ideas on how to fruitfully extend this work.

2. Related Work

Many researchers have tackled problems related to the automation of cameras in 3D environments and the application of cinematographic principles to their operation. Operations that are trivial to carry out in the real world, such as positioning the camera in order to provide an optimal view of some object, can prove to be difficult to automate in the virtual world. Halper & Olivier [9] tackle this problem and suggest the use of a genetic algorithm to evolve increasingly good solutions that gradually converge towards the optimal shot. Such an approach can be used to generate a camera placement that places an object at a particular point in the viewport. The problem becomes more difficult if the subject of interest is moving and the camera has to track it correctly. Halper et al [10] present a camera engine that is designed to

cope with these situations. Their system controls the camera according to a given set of geometric constraints, such as position and orientation of the camera with respect to the subject, and degrees of freedom of the camera. They solve for these constraints on a frame-by-frame basis and try and ensure frame coherence by looking ahead to predict where the camera and the subject are likely to be a few frames into the future.

Explicit attempts to incorporate cinematography into the operation of virtual cameras have tended to be based on film *idioms* [4]. A film idiom is a standard method of applying a number of shots to a given scene. So for example, a scene involving two people having a conversation in a room would have a standard sequence of shots typically employed – start with a shot showing both protagonists, alternate between over-the-shoulder shots of each speaker, then move on to close-ups. Idiom based approaches to virtual cinematography [4][11][1][17][3] store information about idioms and their corresponding shot sequences and cinematic guidelines, and then trigger the use of the correct idiom based on what is occurring within the scene. Christianson et al [4] use this principle to generate cinematic depictions of 3D animations but not in real time. The idiom approach is particularly suitable for dealing with interactive narrative [1] and for depicting conversations between characters in virtual worlds [11][3]. Ting-Cheih et al [17] however, explicitly apply this approach to the real time depiction of action in computer games.

Non-idiom based approaches also exist [8][18][7]. Funge [8] proposes a sophisticated way of modeling the cognitive state of artificially intelligent characters within a game and using this information to direct a camera according to their likely behaviour. A related idea is presented by Tomlinson et al [18] and again involves attempting to apply artificial intelligence research to the problem. Friedman and Feldman [7] also abandon the idiom-based approach and present a method of directly encoding cinematic principles into sets of rules that the camera module is forced to adhere to. Their method is not however designed to work in a real-time context

3. Cinematography Principles

We now summarize some basic principles of cinematography paying particular attention to those relevant to our work. Interested readers are directed to texts such as [2] and [15] for a fuller exposition of these concepts. On a film set the *director* chooses the events to be filmed and the *cinematographer* coordinates the camera operators to film these events. The *editor* edits the resulting footage in order to provide a coherent depiction. The resulting motion picture can be seen as comprising of a sequence of *shots*, where each shot is a continuous view filmed by one camera without interruption. Transitions between these shots are known as *cuts* and the combination of a number of shots within a single setting is called a *scene*. The cinematographer therefore has to make a number of decisions:

- For each scene, an appropriate sequence or arrangement of shots has to be selected. This choice might, for example, be informed by reference to the standard idioms referred to in section 2.

- For each shot, a number of factors pertaining to camera positioning, configuration and placement have to be considered. We elaborate on these factors below.
- For each cut, a method of maintaining *continuity* has to be adhered to. This implies that the transition from one shot to another does not disorientate the viewer.

If we consider a typical FPS game we see that it is significantly simpler in this respect than a film. A game will generally consist of a number of *levels* and these are roughly analogous to scenes in a film. Each of these levels however is comprised of one single continuous shot, and therefore there are no cuts. Our game cinematography system converts this long continuous shot into a series of shots, with cuts generated in real time as play progresses.

Fig. 1. Medium shot (left) and full shot (right)

As mentioned above, the cinematographer, when designing a particular shot, considers numerous factors. These include the *camera angle*, which controls the level of subjectivity of the shot, and hence either involves the viewer, or detaches the viewer from, the action; the *shot type*, which dictates how much of the setting and the characters in it are included; the *lens height*, which is the vertical position of the lens relative to the character(s); the *subject angle*, which is the angle of the lens relative to the characters; and the *lens type*, which controls factors such as depth-of-field. Of most importance from our point-of-view are the issues of camera angle and shot type.

Camera angle is used to determine dramatic impact; with *subjective* camera angles designed to draw the viewer into the scene and provide empathy with the characters; and *objective* camera angles more distant from the action, and intended to provide a more general view of proceedings. The most extreme form of subjective camera angle is the *first-person perspective*, where the camera adopts the view of one of the characters in the scene. This is used in FPS games in order to cause the player to directly empathise with the character he/she is controlling in the game. It is used less often in film because it produces jarring effects for the viewers when other characters look directly at the main protagonist, and hence directly into the camera.

Cinematography advises that the choices of how to combine these different shots to film a scene should be informed by the type of action being portrayed. In particular this hinges on whether the action is controllable or uncontrollable. In our case we are obviously dealing with uncontrollable action and a typical approach in this case would be to employ the *principle of introductions*. This means that a scene is introduced with an *establishing shot* that gives the viewer a general view. It will include the setting of the scene, for example a room or a street, and sets out the

locations of the characters. Now that the viewer is familiar with the setting *character shots* are then employed to introduce the characters. The *full shot* and *medium shot* are both used to frame one or a number of characters. A full shot depicts the entire subject and a medium shot frames the subject from the waist up (see Fig.1). There are also various forms of *close-up shots* that can serve to isolate a portion of the scene, add dramatic emphasis to a part of the narrative, or portray a character's expression. At this point the job of the cinematographer is to portray the action occurring in the scene and various conventions exist for portraying specific actions or activities.

Maintaining continuity between shots is an important consideration for the cinematographer. Without continuity the action can become confusing and frustrating for the viewer. Of course, in standard FPS games continuity is not an issue due to the use of long continuous shots, but if we wish to introduce the notion of cutting between different shots then we have to take it into account. The concept of *screen direction* refers to direction of movement of characters and props, and the looking direction of characters relative to the screen. Continuity can be maintained by ensuring that the screen direction of the subject is the same from shot to shot.

4. Game Cinematography System

We now describe our game cinematography system aimed at FPS games. In section 3 we introduced the roles of the director, cinematographer and editor on a real-world film set. In addition to these there are camera operators who carry out the physical tasks of moving cameras around, adjusting the focus, and so on. Our system is modeled on all of these roles although we decided to roll the director, cinematographer and editor into one software representation, the `Cinematographer`. This means that our `Cinematographer` takes on the action-selection role of the director and the cutting role of the editor, in addition to coordinating the camera operators. This virtual `Cinematographer` and our virtual camera operators are implemented as modifications of the Quake II game engine [12]. This engine provides a first-person view that is directly controlled by the player, so issues such as good visibility of the scene do not factor in, because it is up to the user to control the camera. Our work moves away from this restricted view by providing a number of different types of shots and selecting the appropriate one to use at a given instant based on (a) the action occurring in the game and (b) cinematography principles. The player is granted more general views of the game world when their avatar enters a new room for example and special shots for combat situations. These shots are combined with cinematic devices including freeze frame and slow motion.

In order to implement any form of virtual cinematography two fundamental problems have to be solved:

- The system must have the ability to acquire sufficient information about the scene in order to make decisions about which shots to use.
- The system must store the cinematic guidelines: camera configurations for shots, the timing of cuts, and so on.

We solve the first problem by providing an interface that feeds information about the current state of the game world to the cinematography system, The system has a set of rules which dictate what shots to use in particular situations and these rules are encoded with a finite state machine (FSM) in a similar fashion to [17]. We did not see the necessity to store film idioms in a hierarchical structure like Christianson et al. [4] and He et al. [11] in order to solve the second problem, but instead, like Friedman & Feldman [7], chose to store finer-grained rules within the structure of the FSM used to represent the action.

Cinematography tells us that when filming any subject matter the nature of the content must be considered. If the action is controllable as is the case with motion pictures, multiple takes may be filmed and only one camera is needed. With FPS computer games on the other hand the action is not controllable. There can only be one take of a given shot so cinematography dictates that multiple cameras must be used to provide the maximum choice of footage for editing decisions. In accordance with this principle we created a number of different types of camera operators suited to different tasks with each one carrying out a different type of shot to maximise the editing potential.

In order to do this it was necessary to expand upon the single type of shot granted by FPS games. The first-person perspective is still considered to be an important shot, but one of a set provided by our system. In order to decide what types of shots to add to the game we considered the guidelines for shooting scenes. The principle of introductions, described in the previous section, offered a universal way of coordinating multiple cameras such that the viewer is granted varying levels of subjectivity, dramatic emphasis where necessary, and a better chance of finding her bearings within a game setting.

Our choice of shot types, and therefore camera operator types, takes into account both this principle and the tasks the player has to carry out in a FPS. The actions of the camera operators must be coordinated in such a way that these considerations are accounted for and we do this by designing a `Cinematographer` module. The `Cinematographer` also takes on part of a film editor's role in deciding which camera is to provide the player's view at each instant. The editing decisions are based on the principle of introductions, considerations related to the tasks the player is attempting to carry out, and of course, editing principles.

4.1 CameraBots

Our camera operators are implemented as invisible *bots* [14] that we call *CameraBots*. These are autonomous entities that can freely move about the game world and are given specific instructions regarding their position and their movement in such a way that they mimic the behaviour of camera operators on a film set. We have created a number of these CameraBots, which we will now describe in turn.

EstablishCam. This CameraBot is responsible for providing establishing shots of the room or setting in which the action is currently occurring. The motivation for this is that the principle of introductions in cinematography would dictate that at the start of a scene an establishing shot is used to orient the viewer in the new location. EstablishCam is therefore invoked when the player enters a new room in the game.

However, if an establishing shot has been recently used, or if an establishing shot has been used before at the same location, then EstablishCam will not be used. The shot itself consists of a wide-angle view of the room with the camera pointing towards the avatar. In order to avoid difficulties with player control the system freezes the action in the game for the duration of the shot. To find a suitable shot we position the camera a specified height above the avatar, facing the avatar, and as far back as we can go. We then test if a clear view of the avatar exists from this position. If not, EstablishCam tries increasingly greater subject angles on alternate sides (see Figure 2). If no clear view is found, an establishing shot cannot be made.

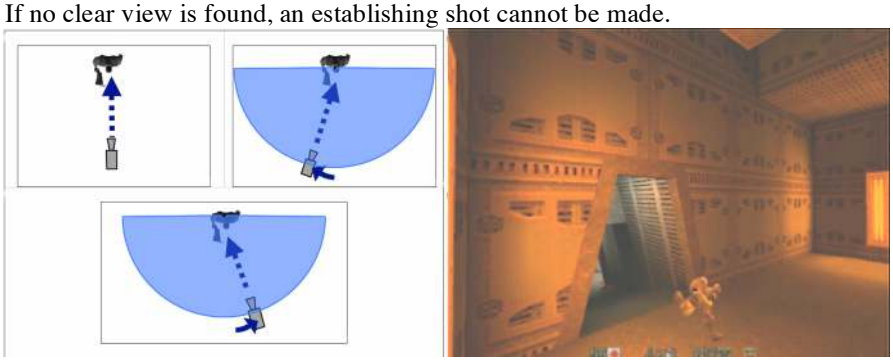

Fig. 2. Left: EstablishCam begins by trying a straight frontal view. If this is not clear it then tries increasingly large deviations from this angle. Right: A typical view provided by EstablishCam.

CharacterCam. This films static full shots of characters occupying the same room as the avatar. The characters being filmed are always enemies and hence it uses a three-quarter low angle to evoke a sense of apprehension. If the full shot is not clear it moves onto the next subject in the room. It tries to ensure that each subject is established, i.e. has enough screen time, before moving on to the next. Once more this is following the principle of introductions and again the action is frozen in the game in order to not put the player at a disadvantage while the character shots are being used.

Fig. 2. View provided by NaviCam

NaviCam. The navigation CameraBot films the game world from behind the avatar's head, looks in the same direction as the avatar, and moves with the avatar to

allow the player to navigate. It is therefore the first of our CameraBots whose movement and positioning, when active, are directly tied to user input. It is similar to the over-the-shoulder camera popular in many games. In cinematography terms this is a subjective camera angle. It is invoked primarily when the user is moving around the game world but not actively engaged in combat. Figure 2 shows the view provided by NaviCam. The familiar problems associated with third-person cameras (e.g. avoiding occlusion and intersecting with walls) are present here also and are solved with standard ray casting techniques.

CombatCam. This CameraBot is designed for combat situations, i.e. shooting at enemies. It provides a first-person perspective that is identical to that usually encountered in FPS games and is directly tied to user input. There are no difficult camera positioning or occlusions issues with this camera.

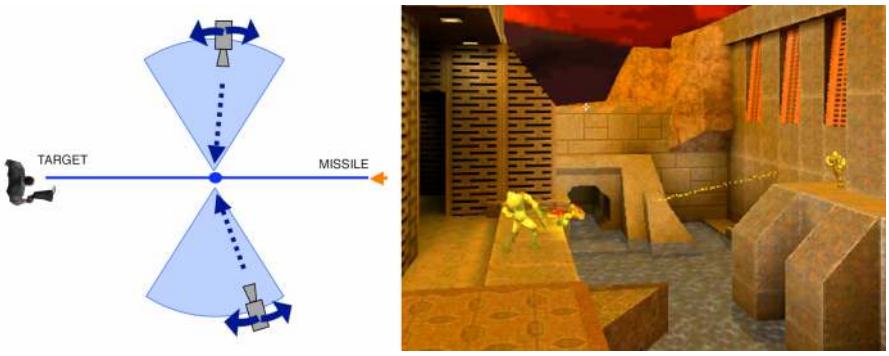

Fig. 3. Left: The positioning method of MissileCam. Right: A typical view provided by MissileCam

MissileCam. The missile CameraBot follows a missile fired by the player if well aimed, and films its effect, i.e. the enemy taking fire. MissileCam's placement method is similar to that of EstablishCam except that it has two subjects – the missile and the target – and it aims at their mid-point. It films from either side of the action axis as defined by the missile's direction of motion and uses a 60º arc on each side to find a suitable placement where it has a clear view of both subjects (see Figure 3). As in the case of the first two CameraBots discussed, the action is frozen while MissileCam is active. A slow-motion mode is then employed to show the trajectory of the missile.

4.2 Cinematographer

The function of the `Cinematographer` module is to create new CameraBots; have them placed in the game world; coordinate them; and select the appropriate one to provide the player's view at each instant. At the beginning of a game the `Cinematographer` creates one of each of the five types of CameraBots discussed above, has these placed in the game world and sets NaviCam to be the player's view.

During each time cycle, the `Cinematographer` assesses the state of the game to decide whether to cut to, and perhaps reposition, a new CameraBot or to continue using the current one (see Figure 4). As already mentioned, it uses the introductions principle from cinematography to inform these decisions. Making introductions in a shooter game, however, requires the use of extra cinematic devices because of the nature of the game.

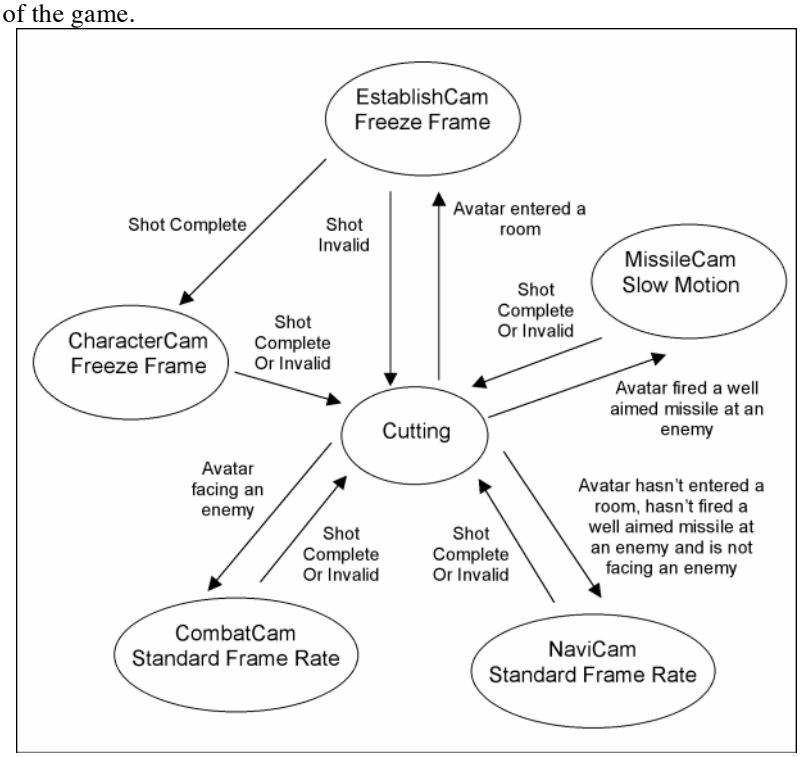

Fig. 4. The `Cinematographer` finite state machine. In addition to the five states corresponding to the CameraBots we define a "cutting" state where editing decisions are made.

In particular, during the game-play within FPS games, and many other genres, the player controls his avatar at all times. This constant association with the avatar poses an obstacle to extending the number of shots provided, since for example, it doesn't make sense for CharacterCam to show the player a shot of another character if he is still controlling the avatar off-screen. Moreover, this particular character might even be shooting the avatar at the time and now the player cannot avoid the attack. Our solution to this problem is to have the `Cinematographer` freeze the current frame in order to introduce the new setting and new characters, i.e. when EstablishCam or CharacterCam are active. When the introductions have been made, the game resumes.

Specifically, in accordance with the introductions principle, if the avatar enters a room it has not occupied before, the `Cinematographer` cuts to EstablishCam and turns on freeze frame; when the room is established it cuts to CharacterCam; when the characters are established it turns off freeze frame and cuts to one of the other three CameraBots. During normal game-play the `Cinematographer` usually activates

NaviCam unless the avatar is facing an enemy in which case it cuts to CombatCam. If the avatar fires at an enemy with a projectile weapon, the `Cinematographer` applies a special shot to this action by turning on slow motion and cutting to MissileCam. Now the player can watch the success or failure of his well-aimed shot in slow motion.

5. Results

The system as described above was implemented as a modification to the Quake II game engine [12]. Our modified version of the engine allows the player to choose to play in normal mode or in our cinematography mode. A key problem that had to be solved in order to create a successful implementation was the provision of a robust method for detecting when the player had entered a new room. This was necessary as a trigger for when to use establishing shots and character shots. An obvious solution to this would have been to manually place markers at room entrances in the Quake level, and then allow the `Cinematographer` to test for the presence of these markers on an ongoing basis. This however would imply that our system would only work on a modified Quake level that has been designed specifically for this purpose. We preferred that the system would be capable of working on any arbitrary Quake level available. For this reason we implemented room detection algorithms within the modified Quake engine we created. This involved using ray casting techniques to test for intersections around the avatar with walls or ceilings and using this information to determine when the avatar has moved from a corridor to a room.

In order to test the suitability of our system to shooter games we had a number of participants play both our modified version of Quake II and the standard version to compare differences between the two in terms of each participant's success in the game. We used three different game levels and made sure that the combinations of levels and versions were evenly distributed. Of a total of 17 participants we tested:

- How our system affected their ability to kill enemies.
- How our system affected their ability to collect items in the game.

The results were as follows. The ratio of kills per hour in our version of Quake II to that in the standard version is 1.26. This means that the participants, on average, killed 26% more enemies per unit time while playing our version. The ratio of pick-ups per hour in our version to that in the standard version is 0.89. This means the participants collected, on average, 11% less items per unit time with our version. A questionnaire was also distributed to participants in order to capture their subjective impressions. On average, players found that the cinematic mode made it a little more difficult to maneuver around the game. However, it allowed them to find their bearings, i.e. to maintain orientation, a little bit better. It also enabled them to find pick-ups more easily, probably because it gives them a broader view of the setting. Players found cuts to be mid-way between natural and confusing. They found the game-play more interesting in cinematic mode and agreed that it was more like

watching a film than playing a game. Most of the participants thought more games should have a virtual cinematography system like ours.

6. Conclusions and Future Work

In this work we have successfully replaced the strict first-person view common to shooter games with a virtual cinematography system that allows more varied camera work by means of a number of different camera modes. Cinematographic rules and principles are used to drive a real-time process by which the system cuts between these different camera modes as play progresses. A limited testing and evaluation process suggests that players find this mode of play to be engaging and interesting, but not to the detriment of game-play to any significant degree. While the system was created as a modification of the Quake II game engine, it was designed with extensibility and portability in mind, and hence adapting it for use in a different context would be a reasonably simple task.

Some obvious extensions and improvements to the system suggest themselves. Firstly the range of events that the `Cinematographer` module is capable of responding to is rather limited (avatar entering a room, characters appearing in a room, avatar facing an enemy and avatar firing weapons). The system could be easily extended to cater for a broader range of events (avatar picking up items, performing leaps, health dropping below a certain level). Secondly a wider arsenal of shot types would provide a more varied and interesting visual experience. Good inclusions in this regard would be panning and tracking shots of various types. Thirdly, no consideration was given to the effects of lighting in the game. This is an important aspect of cinematography and allowing the `Cinematographer` module to interactively control the lighting in the game would open up powerful possibilities. Finally the system does not do a particularly good job of maintaining continuity. Addressing this more seriously would provide definite benefits.

At present the system caters for the player's ongoing interactive view of the game. However, the ideas presented in this paper could also be applied to action replays, spectator views of the game, and level walkthroughs. In each of these cases there is no direct user interaction and hence the problems of striving to not interfere with game-play do not occur. This results in far fewer constraints on what can be done with the camera. We believe a fruitful avenue of further research would be to work on using the system as a means of automatically generating such non-interactive content. The results have the potential be of a cinematic quality rivaling that of cut-scenes.

In conclusion, we believe that in the way that a different set of camera operations is applied to each different genre of motion picture, so too should there be a different set of camera operations, principles and guidelines for the various genres of computer games that exist. The implication of this is that many extensions to our virtual cinematography system would be justified. We also consider the idea of a cinematography module to make a great deal of sense as part of a general-purpose game engine, or perhaps of a middleware solution for game developers. Our reasoning is that it should not be up to a game developer to gain a considerable knowledge of cinematography in order to design a camera system, and it seems

unrealistic to attempt to develop a new virtual cinematography system for each individual game, especially when numerous CameraBot types and editing decisions would most likely be suitable to numerous games, and therefore a general-purpose system would be more appropriate.

References

1 Amerson, D. & Kime, S. Real-Time Cinematic Camera Control for Interactive Narratives. In The Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment, Stanford, CA (2001).

2 Brown, B. Cinematography: Image Making for Cinematographers, Directors and Videographers. Oxford: Focal (2002)

3 Charles, F., Lugrin, J. L., Cavazza, M. and Mead, S. J. Real-Time Camera Control for Interactive Storytelling, 3rd International Conference on Intelligent Games and Simulation, (2002) 73-76

4 Christianson, D. B., Anderson, S. E., He, L., Salesin, D. H., Weld, D. S. & Cohen, M. F. Declarative Camera Control for Automatic Cinematography. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (1996) 148-155

5 Eidos Interactive. Tomb Raider III, computer game for IBM compatible PCs, released by Eidos Interactive, San Francisco, CA (1998)

6 Eidos Interactive. Hitman 2, computer game for IBM compatible PCs, released by Eidos Interactive, San Francisco, CA (2002)

7 Friedman, D. and Feldman, Y. Knowledge-based formalization of cinematic expression and its application to animation, in Proc. EUROGRAPHICS (2002) 163–168

8 Funge, J. Cognitive modeling for computer games. AAAI Spring Symposium on Artificial Intelligence and Computer Games, Stanford University (1999)

9 Halper, N. & Olivier, P. CAMPLAN: A Camera Planning Agent. In Smart Graphics, Papers from the 2000 AAAI Spring Symposium (2000) 92-100

10 Halper, N., Helbing, R. & Strothotte, T. A Camera Engine for Computer Games: Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence. In Proceedings of Eurographics (2001) 174–183

11 He, L., Cohen, M. F. & Salesin, D. H. The Virtual Cinematographer: A Paradigm for Real-time Camera Control and Directing. In Proceedings of SIGGRAPH (1996)

12 id Software. Quake II computer game for IBM compatible PCs, released by id Software, Mesquite, TX (1997)

13 Kneafsey, J. Virtual Cinematography for Computer Games. M.Sc. thesis. Institute of Technology Blanchardstown, Ireland (2006)

14 Laird, J.E. & Duchi, J.C. Creating Human-Like Synthetic Characters with Multiple Skill Levels: A Case Study Using the Soar Quakebot. AAAI tech. report, SS-00-03, AAAI Press, Menlo Park, Calif.. (2000)

15 Mascelli, J. V. The Five C's of Cinematography. Los Angeles: Silman-James Press (1965)

16 Nieuwenhuisen, D. & Overmars, M. H. Motion Planning for Camera Movements in Virtual Environments (2003)

17 Ting-Chieh, L., Zen-Chung, S. & Yu-Ting, T. Cinematic Camera Control in 3D Computer Games. In Proceedings WSCG (2004)

18 Tomlinson, B., Blumberg, B. & Nain, D. Expressive Autonomous Cinematography for Interactive Virtual Environments. In Proceedings of the Fourth International Conference on Autonomous Agents (2000) 317-324.

19 Ubisoft. Prince of Persia: The Sands of Time, computer game for IBM compatible PCs, released by Ubisoft, Austin, TX (2003)