Intelligent Camera Control Using Behavior Trees

Daniel Markowitz, Joseph T. Kider Jr., Alexander Shoulson,
and Norman I. Badler

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389, USA
{idaniel,kiderj,shoulson,badler}@seas.upenn.edu

Abstract. Automatic camera systems produce very basic animations for virtual worlds. Users often view environments through two types of cameras: a camera that they control manually, or a very basic automatic camera that follows their character, minimizing occlusions. Real cinematography features much more variety producing more robust stories. Cameras shoot establishing shots, close-ups, tracking shots, and bird's eye views to enrich a narrative. Camera techniques such as zoom, focus, and depth of field contribute to framing a particular shot. We present an intelligent camera system that automatically positions, pans, tilts, zooms, and tracks events occurring in real-time while obeying traditional standards of cinematography. We design behavior trees that describe how a single intelligent camera might behave from low-level narrative elements assigned by "smart events". Camera actions are formed by hierarchically arranging behavior sub-trees encapsulating nodes that control specific camera semantics. This approach is more modular and particularly reusable for quickly creating complex camera styles and transitions rather then focusing only on visibility. Additionally, our user interface allows a director to provide further camera instructions, such as prioritizing one event over another, drawing a path for the camera to follow, and adjusting camera settings on the fly. We demonstrate our method by placing multiple intelligent cameras in a complicated world with several events and storylines, and illustrate how to produce a well-shot "documentary" of the events constructed in real-time.

Keywords: intelligent cameras, behavior trees, camera control, cinematography, smart events.

1 Introduction

Filmmaking is a complex visual medium that combines cinematography and editing to convey a story. The camera must film the right event at the right time to produce the right picture. Cinematography and editing are both complex art forms in their own right, as they involve many highly subjective stylistic decisions. Given the opportunity to shoot the same event, two different directors are likely to produce very different footage with varying shot choices, shot length, and editing techniques. Unfortunately, many games limit a user's experience to a manually controlled camera, or an automatic camera designed to minimize

J.M. Allbeck and P. Faloutsos (Eds.): MIG 2011, LNCS 7060, pp. 156–167, 2011.

occlusions. This basic control does not use cinematic principles to help tell the game's narrative. Films use more shooting variety and style for capturing movies. Our system provides the user with a way to create and style complex camera transitions to provide a more engaging experience in the game.

Cinematography follows a variety of rules and principles to communicate information and story in a clear and coherent fashion. Christianson and his colleagues [6] describe some of the established cinematographic techniques, and Arijon [2] and Lukas [21] provide additional detail. Cristie and Olivier [8] and Cristie et al. [7] outline the state of the art in camera control in graphics. Some principles include the 180-degree rule: if a horizontal line is imagined to cut across the camera's field of view, the camera should not instantly rotate and cross this line in consecutive shots; if it does, subjects that originally appeared on the right side of the frame will now appear on the left, and vice versa. Similarly, the 30-degree rule states that when the camera cuts to a different view of the same subject, it should be rotated at least 30 degrees in any direction to avoid a jump cut. In the case of clear communication, editing should be essentially "invisible": breaking these rules can lead to jarring edits.

The progression of shots facilitates clear communication: when an event occurs, a so-called "establishing shot" depicts the event from a distance and gives the viewer context; next, close-ups and other shots fill in the details. In single shots, framing assists in highlighting parts of the image that draw attention, by using selective focus or keeping objects in the middle of the frame. Negative space should also be left in front of moving objects, and the actor should always lead. These techniques apply to both live-action and virtual films. These camera semantics are cast as behavior trees in our system. Using behavior trees gives a quick and modular method to create intricate camera styles.

In this paper, we introduce an intelligent camera system that automatically reacts to events that occur in a virtual world. Our camera's actions are programmed as behavior trees that react to "smart events". We capture actions in real-time while following standard principles of cinematography and editing. We provide a user interface for a "director" to control and override camera semantics such as prioritizing one event over another, drawing a path for the camera to follow, and adjusting camera settings on the fly. This allows multiple intelligent cameras to film events, and allows for a customizable approach to editing a final animation. Our test environment tracks multiple rolling balls, a crowd scene, and a conversation to demonstrate how to create a well-shot "documentary" of multiple events edited in real time. Our world is built in the Unity Game engine demonstrating the usefulness of our technique for producing videos for games.

Our system gives the game designer a more robust way to customize how he wants to portray his game. Because the cameras are intelligent, once the camera behavior is programmed, it automatically shoots the character, action, or events occurring in real-time. A designer, much like a film director, brings his own unique style to controlling the player's visual experience. Additionally, our user interface allows players to create custom "cut scenes" out of real-time gameplay, and permits a player to create cinematic replays of his gameplay to post on social media.

2 Background

Many researchers have focused on integrating cinematographic principles into automatic camera control. These approaches provide sophisticated control to a user. Christianson et al. [6] divide cinematography into simple fragments in a "Declarative Camera Control Language". They also develop a "Camera Planning System" that partitions a character's movement into individual actions. He et al. [15] encode a finite state machine that controls the camera's position and parameters. This implementation does not work in real-time and is limited to simple scenarios involving two characters. Our approach works for multiple simultaneous events, operates in real-time, and expands to more robust scenarios, such as crowds, tracking an object/character, and conversations.

The CINEMA system [9] implements a method for camera movements (pans, tilts, rolls, and dollies). This method procedurally generates camera moves based on prior user inputs. CamDroid [10] implements intelligent cameras by defining camera modules similar to Christianson et al.'s [6] approach. While they demonstrate their work on both a conversation and football match, the approach requires prespecified constraints by the user for how to implement a particular shot. Bares et al. [4] developed an interactive fiction system using a constraint-based approach. Bares et al. [3] also looked at using storyboard frames to constrain a virtual camera. Friedman and Feldman [12] used non-monotonic reasoning for their camera's behavior and other knowledge-based systems [13]. Amerson et al. [1] abstractly described cinematographic expressions to control their camera.

Portraying a narrative correctly is an important aspect in storytelling [25], and many games fail to use complex camera behaviors to provide a mood and style. Elson and Riedl [11] introduced Cambot that used an offline algorithm to automatically produce a film. Jhala [18] proposed a method to generate a narrative for events and directives.

Recent camera control work focuses on visibility planning. The camera is automatically controlled to keep the actor visible. Li and Cheng [19] used a probabilistic roadmap to update the camera's position. Halper et al. [14] used a point-based visibility method. Oskam et al. [22] demonstrated a robust global planning algorithm. They used a visibility aware roadmap to avoid occlusions. Of all of the previous camera control systems, Lino et al.'s work [20] is the most similar to our approach. They rely on "Director Volumes" to partition the space, determine visibility cells, and provide details for performing cinematic camera control. The "Director Volumes" allow a user to program his own style by ranking preferences. Our approach does not replace these systems, but complements them. Our contribution provides a dynamic and modular approach to linking various camera behaviors to "smart events". Camera behaviors are determined by the entire sequence of multiple events. A more complex visibility planner can be programmed into our system based on a user's needs. Our demonstration uses a simplified version of the Oskam et al. [22] approach.

Pereira et. al [23] presented a model to describe events stored in a table by their relationships. Their camera control merely frames the active agent, but their system does not respond to events that occur in their scenario. Stocker and

her colleagues [24] programmed behaviors and responses into events. The "smart event" informed an agent how to respond to a scenario. The user does not have to plan and program how agents respond to each other, since the event provides the appropriate behaviors. We leverage this approach in our method to provide a novel control for our intelligent camera system. As new events occur, a camera is assigned to an event, and the event itself provides the necessary parameters to pick and execute a behavior tree to control the camera's semantics. This approach is more efficient for controlling multiple intelligent cameras since it does not spend time planning how to shoot the event.

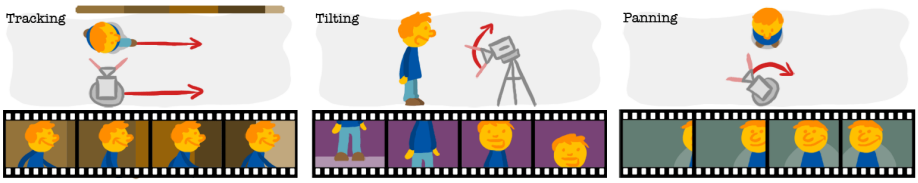

Fig. 1. This figure shows the basic cinematography movement controls for our cameras: tracking, tilting, and panning. Most camera movement is broken down to these controls.

3 Cinematography Principles

Camera shots provide organization and style to a movie or game in a sequence of scenes. Moving the camera, however, is a delicate art based on some well-established principles. Researchers have outlined many complex details of spatio-temporal cinematography standards [2,21]. In a game, however, a user interacts with a virtual world in a sequence of play in real-time. Similar guidelines should be used to guide game cameras as in film. Our intelligent cameras follow similar principles of cinematography to provide a more robust visual experience. This allows a game designer to set various facets of the gameplay's setting, characters, and themes.

Camera Movement: The camera movement shapes and informs a shot's meaning. Basic camera moves include three actions: pans, tilts, and tracks. In general, most complex camera moves are simply mixtures of these basic components. A camera pans when it rotates horizontally, either from left to right or from right to left, normally establishing a scene. The camera rotates vertically to tilt, either from up to down or from down to up. In a tracking shot, the entire camera moves, rather than simply changing the direction of its focus to provide detail. These moves provide 6 degrees of freedom allowing the camera to position and orient itself and are illustrated in Figure 1.

Framing Shots: In games, framing and composition impart information to the player. The tightness of a frame provides different information to the player. If the shot is "tight," it is more personal, focused, and offers less information. This shot creates anxiety in a player. If the shot is "wide" there is more information making it easier for the player to find his objective. Changing the shot type

changes the mood and provides the game with an "emotional evolution". The composition should also allow for empty space in front of the character, both when he is moving and where he is looking (Figure 2). Also, care should be taken not to cut a character off at the ankles, knees or neck.

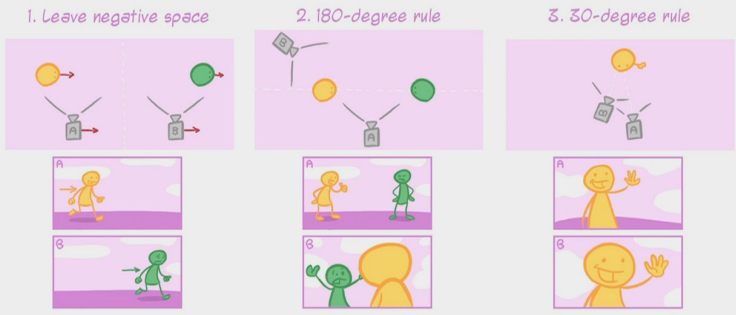

Fig. 2. This figure illustrates some advanced camera shots and editing principles. The top demonstrates how leaving some negative space allows the actor to lead. The middle shows the 180-degree rule related to the invisible line of action in a scene. The bottom displays the 30-degree rule, which shows the minimal distance a camera should move.

Basic Editing Rules: Enforcing basic editing rules provides continuity to a sequence of shots. These act as constraints that ensure that the player is not disoriented in a game. For example, the 180-degree rule is related to the invisible line of action in a scene. Once it's established that the camera is on one side of that line, abruptly cutting across it will be disorienting for viewers. The 30-degree rule ensures the camera moves enough when it cuts, otherwise a jump-cut or indecisive cut results. "Line of action" and "line of interest" are two continuity rules that should be enforced as well [20].

3.1 Intelligent Camera Representation

Using more shot types and enforcing some basic cinematography principles improve a game's narrative, visual appearance, and play. Our intelligent cameras use these three basic camera controls and take care framing shots. We set range parameters on the camera cuts to enforce basic editing rules. We define our intelligent cameras by the following parameters:

- **Position(x,y,z)** - position of the camera
- **Rotation(rx,ry,rz)** - rotation of the camera
- **Zoom(level)** - zoom level
- **DepthOfField(level)** - depth of field
- **FocusPoint(x,y,z)** - focus point if different than center point
- **Style(type)** - behavior style of the camera
- **FollowDistance(dist)** - follow distance
- **FollowHeight(height)** - follow height
- **ShotDuration(time)** - how long the shot will be held

- **Cuttingspeed(speed)** - how fast cuts take place
- **lastCutPosition(x,y,z, rx,ry,rz)** - if only one camera enforces edit rules
- **EventID(ID)** - The smart event the camera is assigned

Shot Idioms: The standard convention in previous work is to break shots up into "fragments" and stitch multiple fragments together, composing an intricate shot idiom as an element in the narrative [6,10,20]. These cinematic shot idioms are sequences of camera setups. An idiom is encoded by defining the necessary parameters in the scene (actors, times, angles). Idioms allow a game designer to bring different styles into frame sequences. The shot idiom is chosen by the "smart event" that assigns a shooting priority to the shots. The idiom's parameters are assigned at run-time for the scene by this "smart event". For example, a simple dialogue scene is handled with a "shot/reverse" shot idiom. The game designer programs these narrative idioms as behavior trees (defined in the next section). So idioms are quickly scripted by the designer and are very modular.

4 Implementation

Behavior Trees are simple data structures that provide a graphical representation and formalization for complex actions. Behavior Trees control our intelligent cameras using "AND/OR" logic with actions, assertions, and other constructions to implement various shot idioms. Behavior Trees are used in modern games to provide agents with intelligence: examples include Halo2 [16], Halo3 [17], and Spore [5]. Behavior trees are preferable to finite state machines for two reasons: goal direction and hierarchical abstraction. A behavior tree can be designed a series of actions geared towards accomplishing a single goal, with contingencies to handle any obstacles that arise. Larger trees built for accomplishing broader goals are built by combining smaller sub-goals, for which subtrees can be invoked.

Figure 3 shows an example of making our intelligent cameras behave like cinematographers. The tree describes how a single camera behaves. At the top of the tree is a "Loop" decorator followed by a "Selector"-in simpler terms, this means that the camera is in a constant loop of "selecting" which of the two subtrees (left or right) to follow. Assertions, the white rectangles in the tree, test something and then return either success or failure. A selector attempts to execute each of its children (in-order), and stops once a child return reports success at its level. If a child fails, the selector advances and tries the next. The left subtree is a behavior for when the camera is not assigned to an event, and the right subtree is behavior for when it is assigned. The Parallel box means that its children execute simultaneously. So, in the left subtree, the Assert node will constantly check to make sure that there's no event assignment, and the camera will "wander" as long as that assertion is returning success. This enforces constraints on the cameras: the parallel node has one child that is constantly enforcing some constraints on the execution of its other child. The abstracted subtree "wander" represents a "Sequence": if you think of: "Selection" acts like "OR" while "Sequence" acts like "AND". Whereas a "Selector" would try children until it finds one that returns success, a "Sequence" node attempts to execute each of its

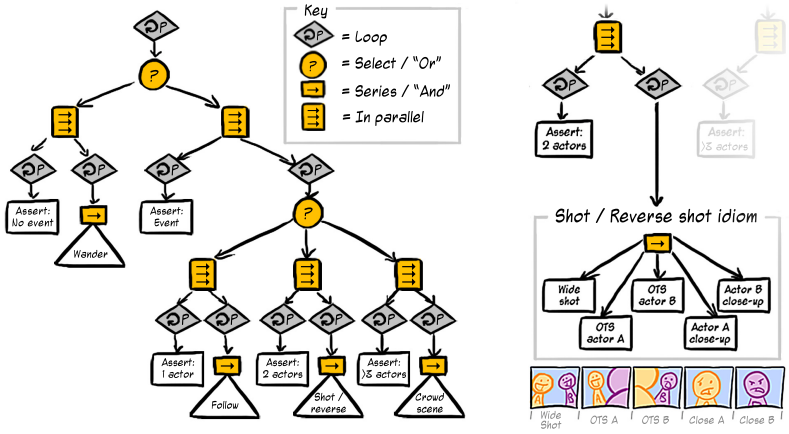

Fig. 3. On the left, this figure shows an intelligent camera's behavior tree. Initially, the camera wanders until it is assigned to an event. The camera reacts to information provided based on the type of "smart event". On the right, we show the abstracted subtree for filming a dialogue scene with a shot / reverse-shot idiom.

children, in order. If any child fails, the sequence node ceases execution (ignoring all subsequent children) and reports failure at its level. The "Sequence" succeeds only after its last child succeeds. In this case, the children of "Wander" would be different actions that cause the camera to move how the designer wishes.

The right subtree starts with another "Parallel" node, which makes sure that it will execute while the camera is still assigned to an event (once the event ends, we should revert back to wandering around). Another "Selector"decides

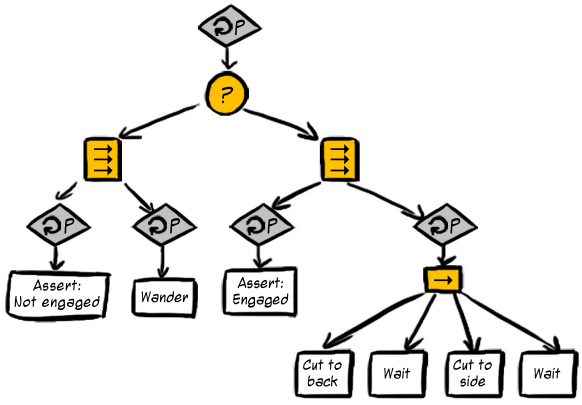

Fig. 4. This figure outlines a simple editing rule: the cameras currently only cut back and forth between two shots: a tracking shot from behind the target, and a stationary shot that pans with the target. This is a diagram of the behavior tree that the cameras are implementing.

what kind of situation we are in. The Selector in the right branch of the tree is the "Cinematographer Node", since it plays the role of cinematographer: deciding what kinds of shots to use based on the current situation.

Figure 4 illustrates a simple editing rule. The cameras currently cut back and forth between two shots: a tracking shot from behind the target, and a stationary shot that pans with the target. The camera has two branches for whether it's currently engaged with an event or not. When the camera is engaged, it enters a shot idiom. In this case, there's only one idiom: cut to a tracking shot behind the target, wait a certain amount of time, cut to a stationary side shot, wait, and repeat (until no longer engaged). This formulation is the foundation for enforcing the cinematography principles discussed above.

4.1 Smart Event Representation

"Smart Events" store the behavioral responses and parameter information inside the event [24]. These events update parameters in the Behavior Trees. This way a designer does not have to program every combination manually and can rely on the event to produce the correct shot. A "message board" updates global information as an event evolves and assigns the intelligent cameras to cover the events. A "blackboard" updates any local information for the behavior tree as the event evolves. Once a "smart event" occurs, it broadcasts to the "message board" and assigns an intelligent camera which is not currently engaged with an event. If all the cameras are assigned, the "message board" looks at the event priority and ShotDuration variables. Our formulation extends the parameters found in Stocker et al. [24] such as "Type", "Position", "Start Time", etc. for the event. Our formulation adds relevant information for the camera such as:

- **ShotIdiom(behavior names)** - set of possible behaviors for the camera
- **CameraIDs(IDs)** - ids of intelligent cameras assigned to event
- **Priority(p)** - the importance of the event
- **Participants(IDs)**- list of actors/objects camera should track
- **Position(x,y,z)** - position of the camera
- **Rotation(rx,ry,rz)** - rotation of the camera
- **Zoom(level)** - zoom level
- **DepthOfField(level)** - depth of field
- **FocusPoint(x,y,z)** - focus point, if different then center point
- **Style(type)** - behavior style of the camera
- **FollowDistance(dist)** - follow distance
- **FollowHeight(height)** - follow height
- **ShotDuration(time)** - how long the shot will be held
- **Cuttingspeed(speed)** - how fast cuts take place
- **LastCutPosition(x,y,z, rx,ry,rz)** - for the cameras

5 Results

Figure 5 shows a screenshot of one of our test environments featuring 3 intelligent cameras in our GUI. The cameras are color-coded across the top so that the user can easily match the physical cameras to their displays. The far left

camera is the "master track" display for the final output color-coded in yellow. The main GUI window can switch among any intelligent camera, the master track, or a user-controlled 3D view. The user-control view allows for a "director" mode, where camera properties can be overridden and views selected, to achieve a novel playback. This is especially useful for making video playbacks of a player's gameplay for posterity. Normally, the intelligent cameras and smart events automatically create the master track with no additional user intervention. The system automatically assigns the intelligent camera and shot idiom behavior based on information from the "smart event". This mode is very useful during real-time gameplay.

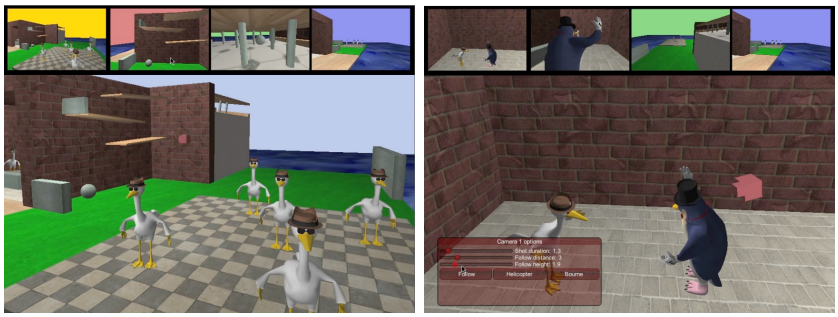

Fig. 5. The figure shows the director GUI. The main window features a user-controlled camera. The yellow window across the top is the "master track" window. The other windows (red, green, blue) are color-coded intelligent cameras. The image on the left is an example of our system shooting various events in a crowd. On the right, we demonstrate a simple dialogue scene when a penguin walks into a pub.

In Figure 5 on the left, the red camera is filming a ball rolling down several slopes (it is currently shooting an establishing shot of the ball with relation to the slopes), while the green camera is following a ball through a complex environment (it is currently shooting a tracking shot behind the ball). The master track is set to display the perspective of the main (user-controlled) camera. The blue camera is not engaged, so it randomly scans the environment until another "smart event" occurs. Figure 5 (on the right) depicts a dialogue scene between a penguin and stork. The red camera is engaged with the event and currently shooting an over-the-shoulder shot within the dialogue idiom. The green and blue cameras are not engaged and currently shooting establishing context shots.

Figure 6 shows a screenshot of a second test environment tracking balls rolling down various slopes. When a ball start rolling it, triggers a "smart event" and assigns a camera to start tracking the motion. The red camera's options are displayed, where the user can adjust certain settings, or change the "genre" of the camera: setting it to "Bourne/24" mode will cause it to shake and cut often. Shot duration sets the length of each shot before an edit occurs (when the shot duration is set to 0, either by the event or user, the camera never cuts from the event until it is fully completed). The camera keeps the ball visible through the best path it can calculate when following the ball through the obstacles.

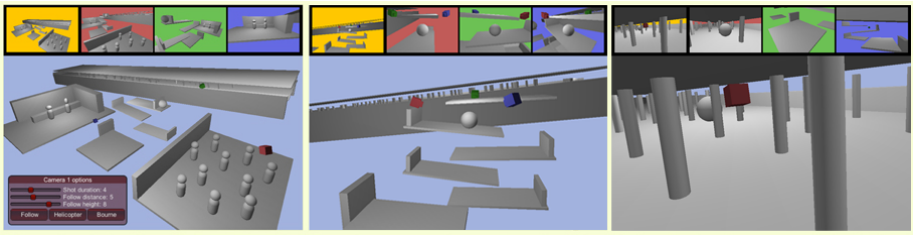

Fig. 6. The figure shows the application of our intelligent camera system in a second virtual world featuring multiple rolling balls and difficult obstacles. The camera follows various shot idioms, while keeping the ball visible. The settings for the red camera are displayed in the left image. Users may use the GUI to compose playbacks of gameplay.

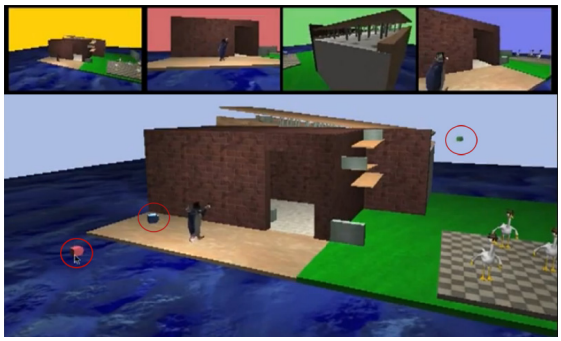

Fig. 7. This figure demonstrates how multiple intelligent cameras are assigned to the scene (highlighted by red circles) to handle the "conversation event" that is unfolding. A director may also assign cameras to enhance the visual experience.

Figure 7 shows a close-up of a "follow" idiom where the event is assigning intelligent cameras to cover the event as the user controls the penguin. These cameras are highlighted in red. Here, the camera is pulled back to "establish" the shot, providing information about the environment. As the shot sequence progresses, the camera draws closer when the player engages another agent. We have tested our scene with multiple events and multiple cameras without producing any lag in gameplay. Our test environments and user interactions use the Unity Game engine, however any game engine would work.

6 Conclusions

In this paper, we have presented a novel intelligent camera system using behavior trees and smart events. Our system automatically positions, pans, tilts, zooms, and tracks the camera to film events. Our system is very modular and dynamically reacts to "smart events" that occur in the scene in real-time. Our intelligent cameras take advantage of the modular abstraction behavior trees provide. This allows a game designer the ability to quickly create new shot idioms, and efficiently reuse the components to create more complex actions. Using

"smart events" provides the necessary parameters to our cameras, and greatly simplifies having to program every possible event manually. This generalizes our approach since the designer does not have to plan out the camera in a game and can instead focus on the game's style and narrative. The intelligent camera will react to the event automatically and film it appropriately. This provides a high-quality visual experience to the player, thereby turning the game into the visual experience of a movie. We provide a user interface to allow a director or user to create their own videos. This feature may be used in a variety of ways in the future. A user could edit his own visual appearance for the game, or construct very strong playback videos of his game highlights to post online. The advantage here is that the camera can record the event BEFORE the great play, save, accomplishment, etc. is actually achieved. In the future, we hope to add more advanced shooting techniques to the system. Parameterizing and implementing camera blur, saturation, color, and contrast effects would all enhance the visual appearance. We also hope to improve the occlusion detection by applying "Director Volumes" and other complex visibility algorithms or directed paths.

References

1. Amerson, D., Shaun, K., Young, R.M.: Real-time cinematic camera control for interactive narratives. In: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, ACE 2005, pp. 369–369. ACM, New York (2005), http://doi.acm.org/10.1145/1178477.1178552
2. Arijon, D.: Grammar of the Film Language. Communication Arts Books, Hastings House Publishers (1976)
3. Bares, W., McDermott, S., Boudreaux, C., Thainimit, S.: Virtual 3d camera composition from frame constraints. In: Proceedings of the Eighth ACM International Conference on Multimedia, MULTIMEDIA 2000, pp. 177–186. ACM, New York (2000), http://doi.acm.org/10.1145/354384.354463
4. Bares, W.H., Grégoire, J.P., Lester, J.C.: Realtime constraint-based cinematography for complex interactive 3d worlds. In: Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI 1998/IAAI 1998, pp. 1101–1106. American Association for Artificial Intelligence, Menlo Park (1998), http://portal.acm.org/citation.cfm?id=295240.296260
5. Hecker, C., McHugh, L., Dyckho, M.A.,, M.: Three approaches to Halo-style behavior tree ai. In: Game Developers Conference (2007)
6. Christianson, D.B., Anderson, S.E., He, L.w., 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, AAAI 1996, vol. 1, pp. 148–155. AAAI Press (1996), http://portal.acm.org/citation.cfm?id=1892875.1892897
7. Christie, M., Machap, R., Normand, J.-M., Olivier, P., Pickering, J.H.: Virtual Camera Planning: A Survey. In: Butz, A., Fisher, B., Krüger, A., Olivier, P. (eds.) SG 2005. LNCS, vol. 3638, pp. 40–52. Springer, Heidelberg (2005)
8. Christie, M., Olivier, P.: Camera control in computer graphics: models, techniques and applications. In: ACM SIGGRAPH ASIA 2009 Courses, SIGGRAPH ASIA 2009, pp. 3:1–3:197. ACM, New York (2009), http://doi.acm.org/10.1145/1665817.1665820

9. Drucker, S.M., Galyean, T.A., Zeltzer, D.: Cinema: a system for procedural camera movements. In: Proceedings of the 1992 Symposium on Interactive 3D Graphics, I3D 1992, pp. 67–70. ACM, New York (1992), http://doi.acm.org/10.1145/147156.147166
10. Drucker, S.M., Zeltzer, D.: Camdroid: a system for implementing intelligent camera control. In: Proceedings of the 1995 Symposium on Interactive 3D Graphics, I3D 1995, pp. 139–144. ACM, New York (1995), http://doi.acm.org/10.1145/199404.199428
11. Elson, D.K., Riedl, M.O.: A lightweight intelligent virtual cinematography system for machinima production. In: AIIDE, pp. 8–13 (2007)
12. Friedman, D., Feldman, Y.A.: Automated cinematic reasoning about camera behavior. Expert Syst. Appl. 30, 694–704 (2006), http://dx.doi.org/10.1016/j.eswa.2005.07.027
13. Friedman, D.A., Feldman, Y.A.: Knowledge-based cinematography and its applications. In: ECAI, pp. 256–262 (2004)
14. Halper, N., Helbing, R., Strothotte, T.: A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. Computer Graphics Forum 20(3), 174–183 (2001)
15. He, L.w., Cohen, M.F., Salesin, D.H.: The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, pp. 217–224. ACM, New York (1996), http://doi.acm.org/10.1145/237170.237259
16. Isla, D.: Handling complexity in the Halo 2 ai. In: Game Developers Conference, p. 12 (2005)
17. Isla, D.: Halo 3 - building a better battle. In: Game Developers Conference (2008)
18. Jhala, A.: Cinematic Discourse Generation. Ph.D. thesis, North Carolina State University (2009)
19. Li, T.-Y., Cheng, C.-C.: Real-Time Camera Planning for Navigation in Virtual Environments. In: Butz, A., Fisher, B., Krüger, A., Olivier, P., Christie, M. (eds.) SG 2008. LNCS, vol. 5166, pp. 118–129. Springer, Heidelberg (2008)
20. Lino, C., Christie, M., Lamarche, F., Schofield, G., Olivier, P.: A real-time cinematography system for interactive 3d environments. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA 2010, pp. 139–148. Eurographics Association, Aire-la-Ville (2010), http://portal.acm.org/citation.cfm?id=1921427.1921449
21. Lukas, C.: Directing for Film and Television. Anchor Press / Doubleday (1985)
22. Oskam, T., Sumner, R.W., Thuerey, N., Gross, M.: Visibility transition planning for dynamic camera control. In: Proceedings of the 2009 ACM SIGGRAPH/ Eurographics Symposium on Computer Animation, SCA 2009, pp. 55–65. ACM, New York (2009), http://doi.acm.org/10.1145/1599470.1599478
23. Pereira, F., Gelatti, G., Raupp Musse, S.: Intelligent virtual environment and camera control in behavioural simulation. In: Proceedings of XV Brazilian Symposium on Computer Graphics and Image Processing, pp. 365–372 (2002)
24. Stocker, C., Sun, L., Huang, P., Qin, W., Allbeck, J.M., Badler, N.I.: Smart Events and Primed Agents. In: Safonova, A. (ed.) IVA 2010. LNCS, vol. 6356, pp. 15–27. Springer, Heidelberg (2010), http://portal.acm.org/citation.cfm?id=1889075.1889078
25. Young, R.M.: Story and discourse: A bipartite model of narrative generation in virtual worlds. Interaction Studies (2006)