Towards an Extended Declarative Representation for Camera Planning

Thomas W. Price and **R. Michael Young**
North Carolina State University
Department of Computer Science
Raleigh, NC 27606
twprice@ncsu.edu, young@csc.ncsu.edu

Abstract

This paper presents a language by which authors can define cinematic sequences declaratively at a high level, while taking advantage of low level camera planning techniques and some procedural elements.

Introduction

As high-quality graphics capabilities become more available, 3D virtual worlds find increasing applications in entertainment, education and storytelling. In many of these applications, it is necessary to generate cinematic sequences that capture events of importance in the virtual world and convey them to the viewer. An author may wish to define how these sequences are filmed, but the exact state of the virtual world during these events, including the positions of the actors, may not be known beforehand. For example, an author may wish to define the cinematic properties of a cutscene in a video game, but exactly when and where that cutscene is triggered might be determined by the player's actions. A similar need arises in filming narratives with procedurally generated plots, where an author might wish to define how certain types of story events are filmed, but the context of these events, including the actors and their locations, cannot be determined in advance.

To address this, we present a language by which authors can define meaningful shot sequences, or *Idioms*, at a high level, while taking advantage of low level camera placement techniques. Once defined, these Idioms can be interpreted by a camera planning system to film scenes in a dynamic context. Our language builds upon past approaches to this problem (Christianson et al. 1996; He, Cohen, and Salesin 1996; Amerson, Kime, and Young 2005) that define this high level structure declaratively. It also allows for the integration of existing camera placement and path planning techniques, as well as the inclusion of some procedural elements to achieve increased expressivity. These components and their related work are addressed in more detail in the following sections.

Declarative Representation

Declarative representations are often used to define an idiom (or a similar structure), specifically how it is composed of lower-level components. The Declarative Camera Control Language (DCCL) (Christianson et al. 1996), for instance, allows an author to define a simple linear ordering of *camera fragments* to comprise an idiom. THE VIRTUAL CINEMATOGRAPHER (He, Cohen, and Salesin 1996) extends this idea by using a finite state machine (FSM) to allow for more complex transitions between *camera modules* based on the state of the virtual world. The camera fragments or modules in these examples are primitive camera routines which have been hard-coded into the camera planner. These primitives are often parameterized with relevant information, such as which actor to film, e.g. `closeUp(actor1)`. The FILM system (Amerson, Kime, and Young 2005) uses a similar declarative structure; however, FILM's low-level components are defined as sets of constraints on the camera's placement to be solved by the camera planner at runtime.

Our language encapsulates these low-level components into a single concept called a *Shot*, similar to the way Drucker and Zeltzer (1995) use generic *camera modules*. We define a Shot as any function which outputs a Virtual Camera (VC), as defined by its position, orientation and field of view. In addition, a Shot may take as input any number of parameters, such as which actor to film or constraints on how the shot should be composed. A Shot could represent a hard-coded routine or a complex constraint solving system. Just as DCCL and the VIRTUAL CINEMATOGRAPHER provide a list of available primitives to the author, a camera planner implementing our language would also provide a set of available Shots. Simple examples of Shots include:

- `LookAt(p1, p2)`: positions the VC at position p1 and orients it towards position p2.
- `ApexShot(a1, a2)`: places the VC such that actor a1 is centered on one side of the screen, and actor a2 is centered on the other side.

We further define a *Transition* to be a specialized Shot which takes as input two VCs and a percentage, $p \in [0,1]$. The function interpolates between the two cameras, transitioning from one to the other over the value of p. For example:

- `Cut(c1, c2, p)`: returns c1 if $p < 0.5$; otherwise returns c2

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

- `SimplePan(c1, c2, p)`: starts at c1 and linearly interpolates that camera's orientation to that of c2 (assumes c1 and c2 share a position).

Our language defines the *Idiom* itself in two parts. First, the author declares a list of VCs, defined using the available Shot functions. Then the author defines a FSM, similar to that found in the VIRTUAL CINEMATOGRAPHER, in which each VC declared earlier is a state, and the VCs are connected by Transitions, triggered by changes in the virtual world. A Transition might be triggered, for instance, when an actor begins talking, when a given amount of time has elapsed, or when an actor becomes occluded. The underlying camera planner would need to disclose available Transition triggers, as it does with available Shots. Like Shots, Idioms may be parameterized, taking inputs such as which actor(s) to film.

Integrating Existing Techniques

Our definition of a Shot is quite broad, and this is a purposeful attempt to allow authors to utilize a wider variety of existing camera planning techniques in their Idioms. For example, many camera planning systems have addressed the problem of positioning a VC to achieve a desired shot composition. These systems often use a set of constraints to define the camera's position, which might include onscreen properties of a character, such as framing (medium or close-up), orientation (front or profile) and occlusion. A typical approach is to translate these properties into a set of mathematical constraints, which are then optimized to yield the camera's position and orientation.

CONSTRAINTCAM (Bares and Lester 1998) employs this technique with a visualization interface that allows the user to request shots with specified "viewing goals" that constrain the camera's placement. The CAMPLAN system (Halper and Olivier 2000) can compute a single shot based on a variety of more complex constraints, including relative constraints on the positions or sizes of two objects onscreen. An alternate approach calculates *Semantic Volumes* (Christie and Normand 2005) that define regions where given camera constraints are satisfied, rather than finding a single ideal shot, though further refinement can be used to select a single camera shot from these volumes. Systems like these could easily be reimagined as Shot functions, taking constrains as inputs, and providing a well-positioned and well-composed VC as output.

An Example

Suppose an author would like to construct an Idiom for filming a conversation between a father and a son. The scene should begin with an apex shot to establish the characters (Shot A), then cut to the father after two seconds (Shot B), and finally cut to the son (Shot C) when he begins talking. If the conversation goes on, the camera should continue to cut back and forth between shots B and C. Further, in order to emphasize the status of the father, he should be shown from a lower angle, and the son should be shown from a higher angle. Let us assume that the camera planner provides a primitive `ApexShot` function as described earlier, as well as a `SolveShot` function, which films a single actor and can be provided with additional compositional constraints, similar to CONSTRAINTCAM or CAMPLAN. With these Shots defined, we can create the Conversation Idiom as shown in Figure 1.

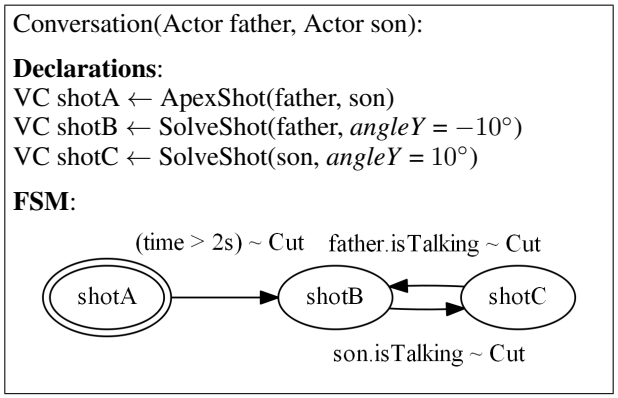

Figure 1: The Conversation Idiom uses both primitive and constraint-solving Shots.

The Idiom first declares three VCs representing shots A, B and C, and defines them using the `ApexShot` and `SolveShot` functions. Special constraint parameters are passed to the `SolveShot` function to achieve the desired vertical viewing angle. The transitions are then defined using a FSM, where the edges are labeled with both the condition for transitioning and the Transition function to use.

Procedural Elements

As with other declarative structures, Idioms are designed to allow authors to the define desired properties of a shot sequence, rather than how those properties are achieved by the camera planner. However, sometimes the increased expressivity offered by a procedural representation can be desirable. To address this, our language allows authors to declare additional variables in the declaration section of an Idiom and pass these variables as arguments to Shot functions. As Pickering and Olivier (2003) suggest, we incorporate variable typing into our representation and allow authors to reference properties of variables. For instance, a VC is composed of a position, orientation and field of view, so if an author has declared a VC variable called `cam`, the author could also reference `cam.position`. In this paper, we will not attempt to enumerate all data types the language should support, but our examples will make references to Actors, VCs and Positions.

As an example of the utility of this convention, consider the following situation: An author would like to create an Idiom representation for a "whip pan," a shot which portrays two actors side by side, starting with a view of one actor (Shot A), and then panning quickly over to the second actor (Shot B). The position of the camera remains fixed. Using existing Shot functions, an author could calculate the desired VC for both Shot A and Shot B, but there would be no guarantee that these shots would share a position, as required for a whip pan. Alternately, one could define a new

Shot specifically for a whip pan, but this might be a labor-intensive process and would involve modifying the camera planner. However, using the `LookAt` and `ApexShot` functions defined earlier, we can construct a whip pan Idiom using our language, as shown in Figure 2.

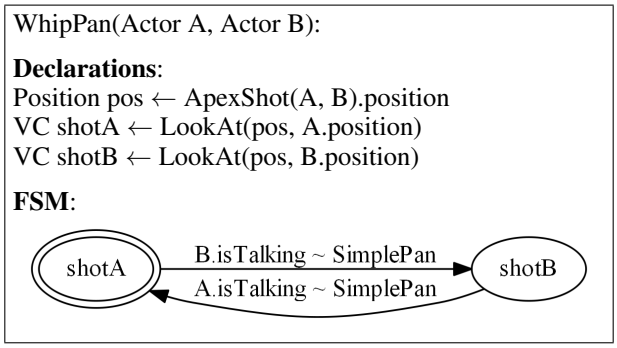

Figure 2: The WhipPan Idiom uses additional variable declarations to create two shots that share a position.

The WhipPan Idiom defines *pos* as a shared camera position for Shots A and B by taking the position of an apex shot of both actors. If our `ApexShot` function is well defined, it will provide a clear view of both actors. For simplicity, Shots A and B are defined by a `LookAt` shot, with the camera positioned at *pos*, filming the respective actor. The FSM starts the sequence with Shot A, and then pans to Shot B when actor B starts talking. Later it can pan back to Shot A when actor A is talking again.

Execution

Our language does not explicitly define how or when an Idiom should be used, except for the requirement that an Idiom's parameters must be specified; however, it is assumed that an external system will be reasoning about this these choices. For instance, a video game engine might start filming an Idiom associated with a specific cutscene when that cutscene is triggered. Alternately, in the context of procedural narrative, an author might annotate Idioms with relevant properties, so that a visual discourse planner (e.g. Jhala and Young 2010) could select an appropriate Idiom to film a scene. Once an Idiom is selected, the camera planner is responsible for evaluating the current Shot function to determine the placement of the camera. It is also responsible for triggering appropriate Transitions in the FSM.

Note that the language makes no commitment as to whether the camera planner runs in real-time or not. If any of the Shots or Transitions employed in an Idiom require off-line calculation, the Idiom will as well, but this should not change how the Idiom is declared. Similarly, any information required by a Shot for calculation, such as the scene geometry and the positions of possible occluders, would be required by the camera planner implementing it. We do not consider this additional information to be a *parameter* of the Shot because it should be provided by the camera planner at runtime and not the author at design time. The author of an Idiom need not be concerned with how the camera planner implements Shots, so long as it is clear which Shots are available. In fact, a camera planner could be continually updated to provide new Shots, Transitions and Transition triggers to authors, without changing the overall structure of the language.

Conclusions and Future Work

The language presented in this paper attempts to take a step towards integrating lower-level camera planning techniques and procedural elements into a declarative representation of an Idiom. However, there are some issues that future work will need to address before the language can be implemented in a camera planner and evaluated. A formal syntax and list of data types will help to define the language more precisely, and viable Shots and Transitions will need to be identified. Additional consideration should also be given to the role of time in the execution of an Idiom. For instance, should the current Shot be evaluated at every frame? If so, would Shots then be required to be continuous functions over time? Further investigation is needed to answer these questions.

References

Amerson, D.; Kime, S.; and Young, R. M. 2005. Real-time cinematic camera control for interactive narratives. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. ACM.

Bares, W. H., and Lester, J. C. 1998. Intelligent multi-shot visualization interfaces for dynamic 3D worlds. In *Proceedings of the 4th International Conference on Intelligent User Interfaces*, 119–126. ACM.

Christianson, D. B.; Anderson, S. E.; He, L.; Salesin, D. H.; Weld, D. S.; and Cohen, M. F. 1996. Declarative camera control for automatic cinematography. In *Proceedings of the Conference of the American Association for Artificial Intelligence*, 148–155.

Christie, M., and Normand, J.-M. 2005. A semantic space partitioning approach to virtual camera composition. *Computer Graphics Forum* 24:247–256.

Drucker, S., and Zeltzer, D. 1995. CamDroid: A System for Implementing Intelligent Camera Control. In *Proceedings of the 1995 symposium on Interactive 3D graphics*.

Halper, N., and Olivier, P. 2000. Camplan: A camera planning agent. In *Smart Graphics 2000 AAAI Spring Symposium*, 92–100.

He, L.; Cohen, M. F.; and Salesin, D. H. 1996. 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*, 217–224. ACM.

Jhala, A., and Young, R. M. 2010. Cinematic visual discourse: Representation, generation, and evaluation. *IEEE Transactions on Computational Intelligence and AI in Games* 2(2):69–81.

Pickering, J., and Olivier, P. 2003. Declarative Camera Planning Roles and Requirements. In *Proceedings of the Third International Symposium on Smart Graphics*.