Firebolt: A System for Automated Low-Level Cinematic Narrative Realization

Brandon R. Thorne[1], David R. Winer[2], Camille Barot[3], and R. Michael Young[2]

[1] North Carolina State University, Raleigh, NC
brthorne@ncsu.edu
[2] University of Utah, Salt Lake City, UT
drwiner@cs.utah.edu, young@eae.utah.edu
[3] Independent Scholar
camille.barot@gmail.com

Abstract. Creation of machine generated cinematics currently requires a significant amount of human author time or manually coding domain operators such that they may be realized by a rendering system. We present FireBolt, an automated cinematic realization system based on a declarative knowledge representation that supports both human and machine authoring of cinematics with reduced authorship and engineering task loads.

Keywords: intelligent cinematography · machinima · tools and systems

1 Introduction

Machinima is a burgeoning field within cinematography, expanding from in-game replays to pre-rendered cinematics in recent years. To support their customers, game studios and other software vendors have created increasingly expressive tools for using game engine technology to orchestrate and render scenes. These cinematic sequencers, such as Valve's Source Filmmaker [19], offer a rich graphical user interface for the construction of virtual scenes, coordination of the actions of virtual actors, and control of virtual cameras used to film the scene. Because all aspects of the production a cinematic – timing, animation, and filming – require human specification and authoring, using these cinematic sequencers is a labor intensive process, potentially requiring many hours of a user's time to be spent in the generation of a single minute of rendered cinematic.

One limitation of typical cinematic sequencers is a design that requires a user to specify low-level details of all story action and camera shot specifications. From a narrative theoretic perspective, a user must specify all aspects of both story and discourse [6], where story consists of the events of a narrative and all the settings, objects and characters involved in their occurrence, and discourse consists of all medium-specific resources used to convey story elements to a viewer. In the approach of conventional cinematic sequencers, story and discourse are conflated, as the user must work to manage their own design in the filming environment. Further, every aspect of story-world behavior must be provided by hand. Similarly, every aspect of the shots that film the world must also be specified by a human.

In contrast, we present a cinematic sequencer that provides an API and uses a well-defined declarative approach to the specification of cinematic narrative that allows systems to decouple the means of production of a sequence from the realization of the sequence. We also design our API to support a range of use cases, including a) human-driven tools that support lower expertise levels than other approaches, and b) the potential for integration with intelligent story and discourse generation tools, to support both automatic production and mixed-initiative interaction. The work presented here is motivated by the lack of declaratively driven realization/rendering engines. We wish to enable authors to easily script story and camera sequences without writing custom code for the rendering engine. A declarative representation that parses the story world and setting from the plan of narration would benefit narrative systems with different underlying structural representation of story, and enables reuse of authored content.

2 Related Work

We will review four categories of research related to the work presented here. First, there are narrative generation systems whose rendering capabilities are tightly coupled to those of generation [15, 17, 11]. Though the thrust of these works is on the determination of content and organization for cinematics, because there was no suitable system available for rendering at the time these tools were built, each employs custom code for rendering. Second, there is a class of systems focusing on geometric camera placement [8]. There are a host of issues addressed in this area of the literature from simple camera placement and target acquisition [5] to computational application of compositional techniques [1]. Our system incorporates simple and serviceable placement techniques in favor of real-time rendering but allows for future extension into more intricate placement algorithms. Third, there are a number of systems supporting cinematography-friendly declarative representation (e.g., [7, 3, 13]). This declarative format will benefit development of training tools for cinematography, allowing users to easily employ visual techniques recommended by such studies as [18]. Experimental design will benefit from a concisely expressive way to analyze existing films which can then be recapitulated in a controlled manner for testing the effects of film editing on comprehension. Finally, there are works which attempt to provide computationally-amenable representations for narrative-based filmmaking such as [14] and MSML [20], whose EventSync architecture we incorporate here.

3 The FireBolt System

The FireBolt System is a fully scriptable cinematic realization system for rendering cinematic visualization in a virtual environment. The system receives input data from a set of declarative representations specifying 1) available actor/object models and animations, 2) a story consisting of actor/object positioning and animations to play at specific time points, and 3) a camera plan consisting of camera shots described using cinematography-based properties and actors/objects in the story.

FireBolt's *Cinematic Model* is a declarative representation for specifying two major narrative units for cinematic generation: actors and domain actions.

3.1 Actors

FireBolt distinguishes an *actor*, which is a role in a story, from a *character model* which is a specific asset used to render the character, including the character's skeleton, mesh, textures, body motions, face, etc. The Cinematic Model is the declarative specifications for the mappings between actor names and character models.

An animation describes a file and a set of animation indices (i.e. temporal offsets relative to the beginning of the clip that are notable time points in the animation). When defining operations to perform in FireBolt when a particular story action has occurred, multiple operations may be coordinated in time using the animation indices. This coordination strategy is similar to that used in the EventSync model in MovieScript Markup Language [20].

Definition 1 (Timepoint). *A timepoint is a whole number corresponding to a time in milliseconds.*

Definition 2 (Animation Indices). *An animation index is a tuple $\langle \lambda, \omega \rangle$ where λ is an index label and ω is a timepoint.*

Definition 3 (Animation Clip). *An animation is a tuple $\langle \eta, \Lambda \rangle$ where η is a file containing an animation trace, and Λ is a set of animation indices.*

Definition 4 (Animation Mapping). *An animation mapping is a tuple $\langle \nu, \kappa \rangle$ where ν is a label and κ is an animation clip.*

Definition 5 (Actor). *An actor is a tuple $\langle n, h, \zeta \rangle$ where n is an actor name (used in the story), h is a character model, and ζ is a set of animation mappings.*

Domain Operator A domain operator is a template of behavior for driving an actor in a virtual world in a time-sensitive manner. Domain actions are used by the story to easily chunk the story into meaningful units, siilar to STRIPS style operators [12] but without commitments to preconditions and effects.

Each domain action consists of a set of parameters (e.g. variables for actors and locations), a set of animations for an actor with relevant temporal offsets, and a set of engine-based mechanics such as creating, deleting, rotating, and translating objects/actors in the virtual world.

Definition 6 (Engine Action). *An engine-action is a predicate with n-ary terms describing a temporally indexed instruction for the virtual environment engine. It is a tuple $\langle \alpha, \upsilon, \circ \rangle$ where α is an actor, υ is an instruction for α and $\circ$ is an animation index specifying temporal parameters for υ.*

Definition 7 (Animation Action). *An animation-action is a tuple $\langle \alpha, k, \circ \rangle$ where $\alpha = \langle n, h, \langle \nu, \langle \eta, \Lambda \rangle \rangle \rangle$ is an actor, $k \in \nu$ is an animation clip label, and $\circ \in \Lambda$ is an animation index.*

Definition 8 (Domain Action). *A domain action is a tuple $\langle P, A, E \rangle$ where P is a set of parameters, A is a set of animation-actions, and E is a set of engine-actions.*

Definition 9 (Cinematic Model). *A cinematic model is a tuple $\langle C, A \rangle$ where C is a set of characters and A is a set of domain actions.*

4 Story

The declarative representation of story is expressed in Impulse [10], a formal language for narrative. Impulse augments a STRIPS-style plan representation [12] with the ability to reason over temporal intervals [2] and model a BDI agent architecture [9].

The story is divided into actions that that drive the actors and objects in a time-sensitive manner. Though Impulse is capable of representing intervals of arbitrary types, FireBolt makes the restriction that interval endpoints be defined in whole numbers. This allows FireBolt to make judgments about the implicit relations of time intervals whose endpoint specifications are not identical. The templates for these actions are described below.

Definition 10 (Story Action). *A story action is a tuple $\langle D, V, \tau \rangle$ where $D = \langle P, A, E \rangle$ is a domain operator, V is a set of values for parameters in P, and $\tau = [s, e]$ is an interval bounded by two timepoints s, e such that $s < e$.*

Definition 11 (Story Timeline). *The story timeline is a function $T_s : K \mapsto A$ mapping timepoints in K to sets of actions containing the animation-actions and engine-actions to initialize or update at the timepoint.*

Definition 12 (Story Model). *The story model is a tuple $\langle T_s, A_S \rangle$ where T_s is a story timeline and A_S is a set of story actions.*

5 Camera Plan

The declarative representation of the camera shots adopts a novel cinematography-friendly shot-description language, *Oshmirto Shot Fragment Language* (OSFL), to specify an expressive but concise array of properties for a shot. The Camera Model packages OSFL descriptions into the discourse structure by defining a total-ordering of shots and their durations, including what story time they should film over.

5.1 Shot Representation

A single shot in cinematography is defined as the film from one cut to the next [4]. However, the camera may take several movements at different times throughout a shot. We wish to both enable a continuation of one movement as well as the initiation of a different movement during a shot. To capture this, we incorporate the notion of a *shot fragment* [7], a temporal interval of a shot in which cinematographic properties are defined. No two shot fragments in the same shot can be overlapping. FireBolt will attempt to film two consecutive shot fragments defined within the same shot without moving the camera between the end of the first and the beginning of the second. If there is only one shot fragment, then that shot fragment constitutes an entire shot.

Algorithm 1 Parse Impulse in Story Model to Timeline T_S

```
1: for each $s_A = \langle D = \langle P, A, E \rangle, V, \tau = [s, e] \rangle$ do
2:     for each timepoint $i$ in $T_s$ from $s$ to $e$ do
3:         for each $a_a = \langle \alpha, k, \langle \lambda, \omega \rangle \rangle \in A$ do
4:             if $s + \omega \leq i < e$ then
5:                 Let $T_s[i] = T_s[i] \cup \{a_a\}$
6:             end if
7:         end for
8:         for each $e_a = \langle \alpha, \upsilon, \langle \lambda, \omega \rangle \rangle \in E$ do
9:             if $s + \omega \leq i < e$ then
10:                Let $T_s[i] = T_s[i] \cup \{e_a\}$
11:            end if
12:        end for
13:    end for
14: end for
```

5.2 Oshmirto Label Specifications

OSFL utilizes a number of label types to convey which of a set of enumerated values each property of camera control is assigned in a given shot fragment. The table below lists the values for each label type. The operationalization for each label type is discussed subsequently.

Additionally, each actor/object in the virtual world, including the camera, has a three dimensional vector corresponding to its position and a three dimensional vector corresponding to its orientation. In some cases, we define the labels in terms of coordinate axes for the sake of expedience. These definitions are relative to a left-handed, y-up, three dimensional Cartesian coordinate system.

Camera Angle : Θ. A camera angle refers to the angle of inclination of the camera on its x axis. Generally the camera is also translated along the y axis relative to the subject in order to acquire the subject in frame. A medium angle indicates that the camera is at the same height of the actor, with no tilt to the camera. A high angle indicates a 30 degree downward angle of the camera oriented towards the actor. A low angle indicates a 30 degree angle upward angle of the camera.

Framing : $\mathcal{F}$. Each actor has a bounding volume about its location that represents an upper bound on the volume occluded by that actor.

Definition 13 (Bounding volume). *A bounding volume b is a polyhedron which circumscribes an actor.*

The framing of an actor indicates a range of acceptable proportions of the height of the bounding volume, b_h, to the height of the screen viewport, v_h.

Direction : $\mathcal{D}$. Direction indicates a range of positions about an actor within which the camera must be placed. This range is based upon the orientation of the actor about the y axis. We enumerate four such ranges: toward, left, right, and away. In effect, the direction specifies the facing of the actor in viewport.

Movement : $\mathcal{M}$. Whenever a change of camera attributes is required during an exposure, a set of movements is used to describe that change. They are $dolly$ (translate the camera in the x-z plane), $crane$ (translate the camera along the y axis), pan (rotate the camera about the y axis), $tilt$ (rotate the camera about the x axis), and $focus$ (set distance at which the camera should focus).

Directive : $\mathcal{X}$. Each movement is associated with a directive, to either move *to* a position, or to move *with* some actor that may be moving in the world. The *to* directive indicates that the argument associated with the movement should be treated as an absolute position or rotation value in world coordinates. For a *pan* movement and a *to* directive, this means that the supplied argument is a heading given by a number of degrees about the y axis. For a *dolly* movement and a *to* directive, the supplied argument is treated as a position where the camera should be at the end of the movement.

The *with* directive indicates that the argument associated with the movement is an object with which the camera should maintain a static relationship. For a *pan* movement using a *with* directive, the camera should rotate about the y axis to keep the argument framed. For a *dolly* movement using a *with* directive, the camera should translate on the x-z plane so that it maintains a fixed distance to the given argument.

Focal Length : $\mathcal{L}$ Focal length is a measure of the optical power of a lens, that is, the distance over which parallel light rays converge as they are brought into focus within the lens. There are several effects of this natural phenomenon, two of which we operationalize in FireBolt. First the focal length affects the expansiveness of the view through the lens. We model this directly as the vertical field of view angle (VFoV) on our virtual camera and use a table of equivalences to relate vertical field of view to commonly available lens focal lengths. Second, the focal length affects the rate at which objects not at the point of focus lose the appearance of focus. This is closely related to aperture, and the calculations are described below.

Aperture : $\mathcal{A}$. Aperture refers to the space through which light enters the camera, which affects both the circle of confusion, used to determine the depth of field (DOF), and luminance of the image. We adopt the canonical f-number aperture labels, but currently only support the DOF effects associated with aperture.

Several optional specifications can be made about a shot fragment. These specifications are formally defined, building up to a formal definition of a shot.

Definition 14 (Duration). *A shot fragment duration is a timepoint specifying the amount of story time over which to map the image captured by a virtual camera to the viewport. It is a pair* (s, e) *where* s, e *are timepoints and* $s < e$.

Definition 15 (Anchor). *An anchor is a position derived from the location of actor dictating an exact world position to place the camera.*

Definition 16 (Angle). *An angle is a tuple* $\langle \theta, \alpha \rangle \mid \theta \in \Theta$ *and* α *is an actor*

Definition 17 (Direction). *A direction is a tuple* $\langle d, \alpha \rangle \mid d \in \mathcal{D}$ *and* α *is an actor*

Definition 18 (Framing). *A framing is a tuple* $\langle f, \alpha \rangle \mid f \in \mathcal{F}$ *and* α *is an actor*

Definition 19 (Static Specification). *A static specification is a grouping of properties defining the positioning of a camera, relative to subjects, at an unspecified timepoint. It is a tuple* $\langle c, \beta, \times, \Upsilon \rangle$ *where* c *is an anchor,* β *is an angle,* $\times$ *is a direction, and* Υ *is a set of framings with unique actors.*

The actor specifications in anchors, directions, and framings may all take unique actors as arguments; however, no two framings can be made on the same actor.

Definition 20 (Movement Specifications). *A movement specification defines an axis with which to translate or rotate the camera, a directive for the movement's target location, and a subject that specifies the target. It is a tuple $\langle \mu, x, \alpha \rangle$ where $\mu \in \mathcal{M}$, $x \in \chi$, and α is an actor.*

Definition 21 (Depth of Field). *A depth of field is a tuple $\langle a, \alpha \rangle \mid a \in \mathcal{A}$ and α is an actor.*

Definition 22 (Shot fragment). *A shot fragment is a tuple $\langle \psi, \Phi, \pi, \ell, \Delta \rangle$ where ψ is a static specification, Φ is a set of movement specifications, π is a depth of field, $\ell \in \mathcal{L}$ is a focal length, and Δ is a duration.*

Definition 23 (Shot). *A shot is a tuple $\langle s, R, \aleph \rangle$ where s is a story time, R is set of shot fragments, and $\aleph$ is a bijection function $\aleph : R \mapsto [1...n] \in \mathbb{N}$ such that $n = |R|$.*

For $r_1, r_2 \in R$, if $\aleph(r_1) = \aleph(r_2) - 1$, then r_1 is filmed immediately before fragment r_2.

Definition 24 (Camera Plan). *A camera plan is a tuple $\langle \mathcal{S}, \Gamma \rangle$ where $\mathcal{S}$ is a set of shots, and Γ is a bijection function $\Gamma : \mathcal{S} \mapsto [1...n] \in \mathbb{N}$ such that $n = |\mathcal{S}|$.*

For $s_1, s_2 \in \mathcal{S}$ if $\Gamma(s_1) = \Gamma(s_2) - 1$, then s_1 is filmed immediately before s_2.

6 Execution

In keeping with the adopted bipartite view of narrative, execution of a set of inputs in FireBolt is performed in two phases: sequencing story actions and filming them. In narratological terms, the sequenced story actions form the story and the filming creates a discourse. Algorithm 1 describes the process for sequencing the story into a form that is executable in the virtual environment. Algorithm 2 describes the process for placing the camera in the virtual world relative to the provided Oshmirto instructions and the story actions to which they relate. The result is a real-time rendered visualization of the specified story through the supplied camera view.

Algorithm 1 begins by iterating over all of the story actions s_A. At each timepoint i in the story timeline T_s which falls between the start and end times $\tau = [s, e]$ of the story action, the animation and engine actions are appended to the set of actions to be invoked at i. The inclusion of the animation and engine actions for a given i is also dependent upon any animation index $\langle \lambda, \omega \rangle$ that may be associated with beginning that action $s + \omega \leq i$, meaning that if this action with its "normal start time", s, and its offset amount, ω, should already have begun by timepoint i, then add the action to the actions that should be executed at i ($T_s[i] \cup a$).

In Algorithm 2 we iterate over each shot $\langle s, R, \aleph \rangle$ in the order given by $\Gamma(S)$ in the camera plan $\langle S, \Gamma \rangle$. For each of the shots, we set the current story time in the virtual world δ to the story time indicated for the beginning of the shot filming. This causes

Algorithm 2 FireBolt Oshmirto Execution Algorithm

```
1: for each $\langle s, R, \aleph \rangle$ in $\Gamma(\mathcal{S})$ of Camera Plan $\langle \mathcal{S}, \Gamma \rangle$ do
2:     Let $\delta = s$
3:     for each $r = \langle \psi, \Phi, \pi, \ell, \Delta \rangle$ in $\aleph(R)$ do
4:         for each $i$ in $T_s$ from $\delta$ to $\delta$+$\Delta$ do
5:             for each $a = \langle \alpha, \diamond, \circ \rangle \in T_s[i]$ do update $a$
6:             end for
7:             if $i = \delta$ then execute $\psi$
8:             else update all $\phi \in \Phi$
9:             end if
10:        end for
11:        Let $\delta$ += $\Delta$
12:        Apply intra-shot transition rules
13:    end for
14:    Inter-shot transition rules
15: end for
```

the story world to be updated to the state effected by the story actions of $T_s[\delta]$. Then for each fragment r in the ordering of shot fragments within the begun shot $\aleph(R)$, step along the timeline $T_s[i]$ from the current story time δ until the end of the shot fragment duration $\delta + \Delta$ is reached. Within each step, update all the story actions within $T_s[i]$, then if this is the first timepoint wherein r is executed, realize the static constraints ψ described in r, otherwise update the movements Φ in r. Once the shot fragment is completed $i = \delta + \Delta$, we move δ to point at the beginning of the next shot fragment. At this point we apply intra-shot transition rules, such as not allowing a new position to be calculated for the camera. Once all the fragments in a given shot have been executed, we move on to the next shot in $\Gamma(S)$ and apply higher level, inter-shot rules such as the $180°$ rule [16].

7 Summary and Potential Future Work

FireBolt is a declaratively-driven cinematic sequencer employing a bipartite model of narrative to inform its execution. It is suitable for use as a rendering system for a range of machinima-producing enterprises from fully generative narrative systems to direct human authorship. In these contexts, FireBolt supports real-time cinematic render performance using commodity hardware.

References

1. Abdullah, R., Christie, M., Schofield, G., Lino, C., Olivier, P.: Advanced composition in virtual camera control. In: Smart Graphics. pp. 13–24. Springer (2011)
2. Allen, J.F., Ferguson, G.: Actions and events in interval temporal logic. Journal of logic and computation **4**(5), 531–579 (1994)
3. Amerson, D., Kime, S., Young, R.M.: Cinematic camera control for interactive narratives. In: Proceedings of the ACM International Conference on Advances in Computer Entertainment. pp. 365–370 (2005)

4. Arijon, D.: Grammar of the film language. Focal Press London (1976)
5. Blinn, J.: Where am i? what am i looking at?(cinematography). Computer Graphics and Applications, IEEE **8**(4), 76–81 (1988)
6. Chatman, S.B.: Story and discourse: Narrative structure in fiction and film. Cornell University Press (1980)
7. 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 American Association for Artificial Intelligent. pp. 148–155 (1996)
8. Christie, M., Machap, R., Normand, J.M., Olivier, P., Pickering, J.: Virtual camera planning: A survey. In: International Symposium on Smart Graphics. pp. 40–52. Springer (2005)
9. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. Artificial Intelligence **42**(2), 213–261 (1990)
10. Eger, M., Barot, C., Young, R.M.: Impulse: a formal characterization of story
11. Elson, D.K., Riedl, M.O.: A lightweight intelligent virtual cinematography system for machinima production. In: Proceedings of the International Conference on AI and Interactive Digital Entertainment. pp. 8–13 (2007)
12. Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2**(3), 189–208 (1972)
13. Halper, N., Olivier, P.: Camplan: A camera planning agent. In: Smart Graphics 2000 AAAI Spring Symposium. pp. 92–100 (2000)
14. Jhala, A.: Exploiting structure and conventions of movie scripts for information retrieval and text mining. In: Proceedings of the 1st Joint International Conference on Interactive Digital Storytelling: Interactive Storytelling, pp. 210–213. Springer (2008)
15. Jhala, A., Young, R.M.: Cinematic visual discourse: Representation, generation, and evaluation. IEEE Transactions on Computational Intelligence and AI in Games **2**(2), 69–81 (2010)
16. Monaco, J.: How to Read a Film: Movies, Media and Beyond. Oxford University Press (2009)
17. Porteous, J., Cavazza, M., Charles, F.: Applying planning to interactive storytelling: Narrative control using state constraints. ACM Transactions on Intelligent Systems Technologies **1**(2), 10:1–10:21 (Dec 2010). https://doi.org/10.1145/1869397.1869399, http://doi.acm.org/10.1145/1869397.1869399
18. Swanson, R., Escoffery, D., Jhala, A.: Learning visual composition preferences from an annotated corpus generated through gameplay. In: IEEE Conference on Computational Intelligence and Games (CIG). pp. 363–370. IEEE (2012)
19. Valve Software: Source FilmMaker. http://www.sourcefilmmaker.com/ (2000–2017)
20. Van Rijsselbergen, D., Van De Keer, B., Verwaest, M., Mannens, E., Van de Walle, R.: Movie script markup language. In: Proceedings of the 9th ACM symposium on Document engineering. pp. 161–170. ACM (2009)