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

Brandon R. Thorne, David R. Winer, Camille Barot and **R. Michael Young**
Liquid Narrative Research Group
Department of Computer Science
Campus Box 8206
NC State University
Raleigh, North Carolina 27695

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 knbowledge representation that supports both human and machine authoring of cinematics with reduced authorship and engineering task loads.

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,[1] 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 (Chatman 1980), 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.

[1] http://www.sourcefilmmaker.com/

In contrast, we present in this paper 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:

- production tools that are not planning-based
- human-driven tools that support lower expertise levels than other approaches
- the potential for integration with intelligent story and discourse generation tools, to support both automatic production and mixed-initiative interaction.

Automated cinematic generation has real world applications in entertainment and education. Blockbuster video games are expensive and time-intensive to make. It is also difficult for amateur directors to test and experiment with directing and cinematography techniques. Cinematics are, in general, an effective way to visualize narrative information.

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.

Related Work

We will review four categories of research related to the work presented here.

First, we point the reader's attention to the need for cinematic rendering tools for virtual worlds. There are narrative generation systems such as Darshak (Jhala and Young 2010) and the Merchant of Venice (Porteous, Cavazza, and Charles 2010) whose rendering capabilities are tightly coupled to those of generation. Though the thrust of these works is generation, there was no suitable system available for rendering without writing custom code. Cambot (Elson and Riedl 2007) also inhabits this generative space but using an iterative approach of actor placement, filming and reel scoring. Here our system could also have been employed to ease the engineering burden.

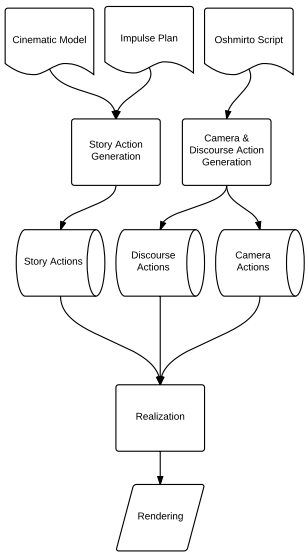

Figure 1: High Level Data Flow of FireBolt Cinematic Generation System

Second, there is a class of systems focusing on geometric camera placement. There are a host of issues addressed in this area of the literature from simple camera placement and target acquisition (Blinn 1988) to computational application of compositional techniques (Abdullah et al. 2011). 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 is a history of supporting cinematography-friendly declarative representation such as fragments (Christianson et al. 1996). This declarative format will benefit development of training tools for cinematography, allowing users to easily employ visual techniques recommended by such studies as (Swanson, Escoffery, and Jhala 2012). 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.

Fourth, there are works which attempt to provide computationally-amenable representations for narrative-based filmmaking such as (Jhala 2008) and MSML (Van Rijsselbergen et al. 2009), whose EventSync architecture we incorporate here.

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.

Cinematic Model

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

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 document containing the declarative specifications for the mappings between actor names and character models. Actors can easily be recast to different models and animation files.

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 given 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 (Van Rijsselbergen et al. 2009).

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, such as STRIPS style operators 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.*

Story

The declarative representation of story adopts an Impulse (Eger, Barot, and Young) representation, a formal language for narrative developed in prior work. Impulse augments a STRIPS-style plan representation (Fikes and Nilsson 1972) with the ability to reason over temporal intervals (Allen and Ferguson 1994) and model BDI agent architecture (Cohen and Levesque 1990). The Impulse language is preferred for FireBolt because of the time-sensitive commitments required for cinematic visualization. The elements used by FireBolt are a set of actions, objects, and sentences. For example, a set of at predicate sentences are used initialize the story world state and place actors at the appropriate positions to begin the story.

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 judgements about the implicit relations of time intervals whose endpoint specifications are not identical. For instance, a valid expression of intervals in Impulse might read as $1 - \zeta$ and $\mathcal{K} - \alpha$, but these intervals give no clue as to their relationship overall.

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.*

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.

Shot Representation

A single shot in cinematography is defined as the film from one cut to the next. 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* (Christianson et al. 1996), 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.

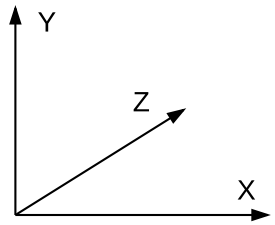

Figure 2: Oshmirto labels are defined with respect to a left-handed, y-up, three dimensional Cartesian coordinate system.

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.

Label Name	Symbol	Set
camera angle	Θ	$\{high, med, low\}$
framing	$\mathcal{F}$	$\{cu, wst, full, long, elong\}$
direction	$\mathcal{D}$	$\{toward, away, left, right\}$
movement	$\mathcal{M}$	$\{dolly, crane, pan, tilt, focus\}$
move directive	$\mathcal{X}$	$\{to, with\}$
focal length	$\mathcal{L}$	$\{28, 35, 50, 100, 150\}$
aperture	$\mathcal{A}$	$\{1.4, 2.8, 4, 5.6, 8, 11, 16, 22\}$

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 (Figure 2).

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. This positioning is described visually in Figure 3.

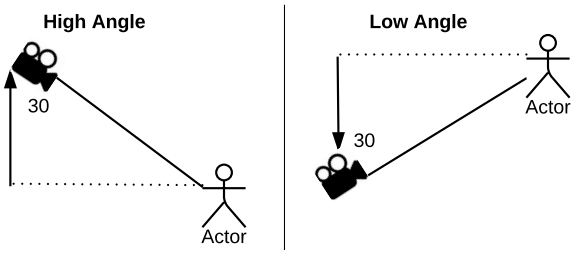

Figure 3: High and low camera angles. The camera is oriented about its x axis then translated on its y axis to frame the subject.

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.

Framing Name	Symbol	$Min[b_h : v_h]$	$Max[b_h : v_h]$
close-up	cu	2.50:1	5 :1
waist	wst	1.75:1	2.50:1
full	$full$	0.75:1	0.95:1
long	$long$	0.50:1	0.75:1
extreme-long	$elong$	0.25:1	0.50:1

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 as shown in Figure 4. 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
$focus$ - 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.

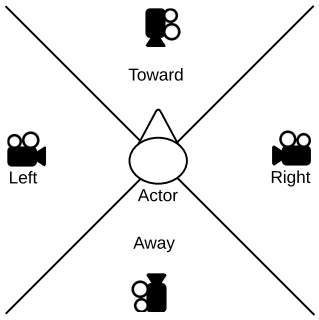

Figure 4: Top-down view of an actor with the space around it divided into four regions into which the camera may be placed. Toward describes the region facing the actor, Right and Left are on the right and left of the actor respectively, and Away denotes the region to the rear of the actor.

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.

Focal Length	VFoV
28mm	28.3°
35mm	21.7°
50mm	15.3°
100mm	7.6°
150mm	4.9°

Second the focal length affects the rate at which objects not at the point of focus lose the appearance of focus. This is intimately related to aperture, and the calculations are described in that section.

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 illuminance of the image. We adopt the canonical f-number aperture labels, but currently only support the DOF effects associated with aperture. Below is a table of f-numbers

and their correlation with acceptable circle of confusion diameters(ACoC-D) in inches given a 50mm focal length and subject ten feet away.

f-Number	ACoC-D	f-Number	ACoC-D
f/1.4	10	f/8	62
f/2.8	21	f/11	91
f/4	30	f/16	153
f/5.6	42	f/22	292

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.

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 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 Parse Impulse in Story Model to Timeline T_S

for each $s_A = \langle D = \langle P, A, E \rangle, V, \tau = [s, e] \rangle$ **do**
  for each timepoint i in T_s from s to e **do**
    for each $a_a = \langle \alpha, k, \langle \lambda, \omega \rangle \rangle \in A$ **do**
      if $s + \omega \leq i < e$ **then**
        Let $T_s[i] = T_s[i] \cup \{a_a\}$
      end if
    end for
    for each $e_a = \langle \alpha, v, \langle \lambda, \omega \rangle \rangle \in E$ **do**
      if $s + \omega \leq i < e$ **then**
        Let $T_s[i] = T_s[i] \cup \{e_a\}$
      end if
    end for
  end for
end for

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 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

Algorithm 2 FireBolt Oshmirto Execution Algorithm

for each $\langle s, R, \aleph \rangle$ in $\Gamma(\mathcal{S})$ of Camera Plan $\langle \mathcal{S}, \Gamma \rangle$ **do**
  Let $\delta = s$
  for each $r = \langle \psi, \Phi, \pi, \ell, \Delta \rangle$ in $\aleph(R)$ **do**
    for each i in T_s from δ to δ+Δ **do**
      for each $a = \langle \alpha, \diamond, \circ \rangle \in T_s[i]$ **do** update a
      end for
      if $i = \delta$ **then** execute ψ
      else update all $\phi \in \Phi$
      end if
    end for
    Let δ += Δ
    Apply intra-shot transition rules
  end for
  Inter-shot transition rules
end for

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.

An Example

Story Action Execution

Figure 5 shows a reduced syntax story model and the domain actions and object mappings required for FireBolt to realize the story. This particular story is one of a containment breach. We can see in the domain actions there is a walk-from-to action that takes an actor, an origin position, and destination position as parameters. FireBolt, upon reading in a walk-from-to action will decompose it into two animations, one for walking to the destination and one for orienting toward it, and two engine actions, one for translating the character and one for rotating it. Similarly the open-door action is decomposed into a button press animation, a door opening animation, a turn animation, and a button light change animation. An engine action is also used to rotate the actor toward the door. The coordination among all of the animations activated by the open-door domain action is accomplished through the use of animation indices on the button press animation. The button press animation starts when the domain action starts and then at the time when the actor's hand is at the button, the button light change animation is triggered. When the button press animation is completed, the door opening and actor turning animations are begun. The actor turning animation's duration is further modified by 500 millisecond maximum duration.

The object mappings section defines which model to use for the given name of the actor as well as which animations to play for a given animation action. The story model lists the story actions that will occur over story timeline T_s. Story actions occur over an interval of 10-12000 with the interval of walkToDoor being contained within that of open1.

Camera Action Execution

Figure 6 shows a reduced syntax version of the Oshmirto camera plan used to film the example scene. In block 1 filming begins at story time 0. The camera is placed at exact (x,z)

Domain Actions

open-door (?a ?b ?d)
anim: (press ?b)
anim: (open ?d)
anim: (turn-to ?a ?d)
anim: (activate ?b)
engine: (rotate-to ?a ?d)

walk-from-to (?a ?o ?d)
anim: (walk ?a)
anim: (turn-to ?a ?d)
engine: (translate ?a ?o ?d)
engine: (rotate-to ?a ?d)

Object Mappings

name: secretAgent
model: "m005.fbx"
press: "finger-poke.fbx"
walk: "slow-walk.fbx"
turn-to: "foot-shuff.fbx"

name: containmentDoor
model: "containDoor.prefab"
open: "open-contain-door.anim"
close: "close-contain-door.anim"

name: doorButton
model: "defaultButton.prefab"
activate: "changeToGreen.anim"
deactivate: "changeToRed.anim"

Story Model

name: walkToButton
action: walk-from-to
?a: secretAgent
?o: (10, 0, 5)
?d: (0.5, 0, 5)
interval: 10 - 4999

name: open1
action: open-door
?a: secretAgent
?b: doorButton
?d: containmentDoor
interval: 5000 - 12000

name: walkToDoor
action: walk-from-to
?a: secretAgent
?o: (0.5, 0, 5)
?d: (0.5, 0, 1)
interval: 7250 - 10000

Figure 5: Story and Domain Specifications, simplified from XML syntax.

coordinates, focal distance is calculated to secretAgent, and a lens is selected to frame secretAgent according the height ratio range for a Full framing. Over the duration of the shot, the camera is rotated about its y axis to track secretAgent, and focus is continually adjusted to keep secretAgent crisp. At the end of 6250 milliseconds, the camera has filmed the entire walkToButton and the button pressing portion of the open1 action.

Fragment 2 of block 1 is a similar specification but requires that the camera be place to the left of secretAgent. Because that constraint connot be satisfied without moving the camera, FireBolt relocates the camera to the center of the room, directly to the left of secretAgent. The camera then pans and focuses with secretAgent as he walks to the door.

Block 2 changes the story time that filming should begin for its shot fragments from what is then current 6250 + 4200 of the previous shot fragment durations + 0 (initial story time) to 9000. To do this, all story actions that were not completed by story time 9000 are undone and reapplied up to their progress at story time 9000. When filming begins for block 2, secretAgent is not quite to his mark at the door, and the door is once again closed.

Fragment 1 of block 2 specifies a lens of 100mm along with a direction of away containmentDoor with a Full framing on the door. Initially FireBolt attempts to place the camera directly behind the door, but the 100mm lens requires a greater distance to the subject to gain a full framing than can be accommodated before attempting to move the camera through the wall behind the door. FireBolt then searches outward radially until it finds a satisfying location in the corner of the room just within the direction boundary for away.

block: 1
storyTime: 0
fragment: 1
  static:
    anchor: (0.25,10)
    focus: secretAgent
    angle: med secretAgent
    framings:
      Full: secretAgent
    movements:
      Pan-with secretAgent
      Focus-with secretAgent
  duration: 6250
fragment: 2
  static:
    direction: left secretAgent
    focus: secretAgent
    angle: medium secretAgent
    framings:
      Full: secretAgent
  movements:
    Pan-with secretAgent
    Focus-with secretAgent
  duration: 4200

block: 2
storyTime: 9000
fragment: 1
  static:
    direction: away containmentDoor
    f-stop: 5.6
    lens: 100mm
    angle: medium secretAgent
    framings:
      Full: containmentDoor
  duration: 4000

Figure 6: Camera Plan simplified from XML syntax, with shot fragment examples.

Evaluation and Potential Future Work

Decoupling

Using a declarative representation for the story and discourse inputs to FireBolt decouples the cinematic realization system from the generative process used to create its inputs, be it computationally or human authored. FireBolt is in use as the cinematic renderer for a story and discourse planning pipeline on the computationally driven end of the spectrum while Impulse and Oshmirto plans are written by hand for cognitive psychological experiments on the other. In between the two extremes, there is a cinematic sequencer, ScreenPlay, with the canonical graphical user interface exploiting FireBolt as its renderer.

Performance

While it is possible to create configurations of actors (generally those with lighting effects embedded prove problematic) that challenge the ability of commodity hardware to allow FireBolt to render a cinematic in real-time, we have found that for scenes with directional lighting only, we are able to support dozens of concurrent actors in story plans of thousands of actions. We take advantage of view frustum culling to avoid rendering animations for actors that will not be seen in the current shot while continuing execution of the underlying animation and engine actions to keep the unseen actor in sync with the visible actors. To reduce expense for re-creating actors that are periodically added and removed in the scene, we make use of a simple pooling strategy to avoid completely rebuilding the actors every time they are needed.

As can be seen in Algorithm 1, the process for translating the story actions into the internal engine and animation actions has a high cyclomatic complexity. This operation performs well for small sets of actors or stories with few actions, but does not scale gracefully, taking over a minute of load time to begin playback for our rendering test mentioned above. This poor scaling could be partially ameliorated by initially loading only a small percentage of the story and then loading the rest concurrently as the cinematic renders. In order to take advantage of such an optimization, however, we would need to impose an additional stricture on the Impulse story plan, namely that the actions appearing in it be ordered with respect to time intervals. Currently we make no such stipulation regarding the format of our story plan input.

Expressivity

Story Actions Because we associate each actor with a combination of model and animations that are unique to it, we are able to have the visual realization of an action being performed by different actors be as idiosyncratic as though it were done by different people. This lends a level of flexibility in the creation of more abstract domain actions than occur in (Porteous, Cavazza, and Charles 2010) where specific actors and even specific animations are directly linked to the domain action.

Using the EventSync model of MSML (Van Rijsselbergen et al. 2009) has provided us with the ability to coordinate multiple actors and animation and engine actions involving those actors by enumerating animation indices for an effecting animation in a domain action. Because we rely on human author annotation to derive this information from the animation clips themselves and currently have no principled method of reasoning about the meanings of the indices, a supplementary approach to coordination is warranted. Additional markup specified for the Cinematic Model could allow us to notate timing relationships among engine and animation actions without the need to have an animation as the effector for a domain action. The engine and animation actions could operate with static offsets or relative percentage completion criteria.

An analogous locational extension to the Cinematic Model would be syntax for the specification of points in the virtual world as being at the same place as an actor parameter at the time the engine action should initialize. This would obviate the need for so many positional parameters as we are currently specifying in the story plan.

Camera Actions Using Oshmirto we are able to describe shots as component fragments that may be combined to form even the most complex sequences of camera movements in use in popular cinematography to date. Not all shot types can be described with a uniform level of difficulty, however. In particular we have encountered several situations in which more types of movement actions would be of use. For example describing an orbit around an actor requires foreknowledge of the position of the actor and subsequent decomposition of the orbit into points that the camera may then $dollyto$. It would be much more expedient from an authoring perspective to introduce into Oshmirto an orbit movement that centers on an actor, taking a radius and an angular velocity to describe this camera movement. Similar shorthand could

be developed for maintaining a position relative to a set of actors or traveling along a trajectory.

Lighting

Currently lighting decisions are made outside of FireBolt. In the simplest case, Firebolt operates on a scene that is lit by a single directional light. Alternatively, objects can be created that emit light such as torches and placed as actors in a scene. In order to more fully express the language of cinematography, we need to incorporate mechanisms for placing and describing lights in our scenes as well as how those lights should be perceived by the camera. The Cinematic Model or a separate Environment Model might incorporate the descriptions of the lights (color, brightness, location, etc) while the perception could be introduced into the Oshmirto aperture implementation (adding the illuminance property that aperture settings influence in real-world cinematography).

Conclusion

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.

Acknowledgments

We thank Thomas E. Ackerman for many discussions regarding the characterization of shots and shot sequences.

This material is based upon work supported in whole or in part with funding from the Laboratory for Analytic Sciences (LAS). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the LAS and/or any agency or entity of the United States Government.

References

Abdullah, R.; Christie, M.; Schofield, G.; Lino, C.; and Olivier, P. 2011. Advanced composition in virtual camera control. In *Smart Graphics*, 13–24. Springer.

Allen, J. F., and Ferguson, G. 1994. Actions and events in interval temporal logic. *Journal of logic and computation* 4(5):531–579.

Blinn, J. 1988. Where am i? what am i looking at?(cinematography). *Computer Graphics and Applications, IEEE* 8(4):76–81.

Chatman, S. B. 1980. *Story and discourse: Narrative structure in fiction and film*. Cornell University Press.

Christianson, D. B.; Anderson, S. E.; He, L.-w.; Salesin, D. H.; Weld, D. S.; and Cohen, M. F. 1996. Declarative camera control for automatic cinematography. In *AAAI/IAAI, Vol. 1*, 148–155.

Cohen, P. R., and Levesque, H. J. 1990. Intention is choice with commitment. *Artificial intelligence* 42(2):213–261.

Eger, M.; Barot, C.; and Young, R. M. Impulse: a formal characterization of story.

Elson, D. K., and Riedl, M. O. 2007. A lightweight intelligent virtual cinematography system for machinima production. In *AIIDE*, 8–13.

Fikes, R. E., and Nilsson, N. J. 1972. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

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

Jhala, A. 2008. Exploiting structure and conventions of movie scripts for information retrieval and text mining. In *Interactive Storytelling*. Springer. 210–213.

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Trans. Intell. Syst. Technol.* 1(2):10:1–10:21.

Swanson, R.; Escoffery, D.; and Jhala, A. 2012. Learning visual composition preferences from an annotated corpus generated through gameplay. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 363–370. IEEE.

Van Rijsselbergen, D.; Van De Keer, B.; Verwaest, M.; Mannens, E.; and Van de Walle, R. 2009. Movie script markup language. In *Proceedings of the 9th ACM symposium on Document engineering*, 161–170. ACM.