to appear AAAI '96

Declarative Camera Control for Automatic Cinematography

David B. Christianson
Sean E. Anderson *Li-wei He*
David H. Salesin *Daniel S. Weld* *Michael F. Cohen**

Department of Computer Science and Engineering
University of Washington
Seattle, Washington 98195

*Microsoft Research
One Microsoft Way
Redmond, WA 98052

{*dbc1,lhe,salesin,weld*}*@cs.washington.edu*, *seander@cs.stanford.edu*, *mcohen@microsoft.com*

Abstract

Animations generated by interactive 3D computer graphics applications are typically portrayed either from a particular character's point of view or from a small set of strategically-placed viewpoints. By ignoring camera placement, such applications fail to realize important storytelling capabilities that have been explored by cinematographers for many years.

In this paper, we describe several of the principles of cinematography and show how they can be formalized into a declarative language, called the *Declarative Camera Control Language* (DCCL). We describe the application of DCCL within the context of a simple interactive video game and argue that DCCL represents cinematic knowledge at the same level of abstraction as expert directors by encoding 16 idioms from a film textbook. These idioms produce compelling animations, as demonstrated on the accompanying videotape.

Introduction

The language of film is a complex one, which has evolved gradually through the efforts of talented filmmakers since the beginning of the century. As a result the rules of film are now so common that they are nearly always taken for granted by audiences; nonetheless, they are every bit as essential as they are invisible. Most interactive 3D computer graphics applications (*e.g.*, virtual chat managers, interactive fiction environments, and videogames) do not exploit established cinematographic techniques. In particular, most computer animations are portrayed either from a particular character's point of view or from a small set of strategically-placed viewpoints. By restricting camera placement, such applications fail to realize the expository capabilities developed by cinematographers over many decades. Unfortunately, while there are several textbooks that contain informal descriptions of numerous rules for filming various types of scenes (Arijon 1976; Lukas 1985; Mascelli 1965), it is difficult to encode this textbook knowledge in a manner that is precise enough for a computer program to manipulate.

In this paper, we describe several of the principles of filmmaking, show how they can be formalized into a declarative language, and then apply this language to the problem of camera control in an interactive video game. Specifically, we describe the *Declarative Camera Control Language* (DCCL) and demonstrate that it is sufficient for encoding many of the heuristics found in a film textbook. We also present a *Camera Planning System* (CPS), which accepts animation traces as input and returns complete camera specifications. The CPS contains a domain-independent compiler that solves DCCL constraints and calculates the dynamical control of the camera, as well as a domain-independent heuristic evaluator that ranks the quality of the candidate shot specifications that the compiler produces. We demonstrate a simple interactive video game that creates simulated animations in response to user input and then feeds these animations to CPS in order to produce complete camera specifications as shown on the accompanying videotape.

Our prototype video game serves as a testbed for applications of DCCL and CPS. However, there are number of alternative applications to which both DCCL and CPS might be applied. Within the realm of video games, Multi-user Dungeons (MUDs), and interactive fiction, automated cinematography would allow an application to convey the subjective impression of a particular character without resorting to point-of-view shots.[1] Because many MUDs operate over long periods of time, an automated cinematography system could provide users with customized summaries of events they had missed while they were away. Alternatively, automated cinematography could be used to create natural interactions with the "intelligent agents" that are likely to take part in the next generation of user interfaces. Automated cinematography could also be used to assist naive users in the creation of desktop videos, or for building animated presentations. In the latter case, Karp and Feiner have shown (Karp & Feiner 1990; 1993) that animated presentations can be effectively designed on a computer, reducing costly human involvement and allowing presentations to be customized for a particular viewer or situation.

[1] Most current games, of which *Doom* is the classic example, still provide each participant with a single point-of-view shot; however, a number of games such as *Alone in the Dark*, *Fade 2 Black* and *Virtua Fighter* have begun to employ a wider variety of perspectives.

Principles of Cinematography

Although a film can be considered to be nothing but a linear sequence of frames, it is often helpful to think of a film as having structure. At the highest level, a film is a sequence of *scenes*, each of which captures some specific situation or action. Each *scene* in the film is composed of one or more *shots*. A single shot covers the small portion of a movie between when a camera is turned on and when it is turned off. Typically, a film is comprised of a large number of individual shots, with each shot's duration lasting from a second or two in length to perhaps tens of seconds.[2]

Camera Placement

Directors specify camera placements relative to the *line of interest*, an imaginary vector connecting two interacting actors, directed along the line of an actor's motion, or oriented in the direction the actor is facing. Figure 1 shows the line formed by two actors facing each other.

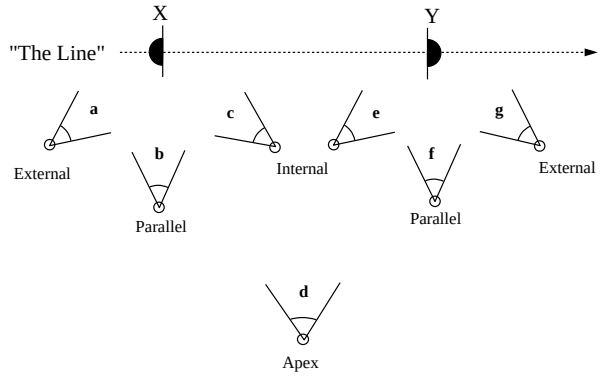

Figure 1: Camera placement is specified relative to the "line of interest." (Adapted from figure 4.11 of (Arijon 1976))

Shooting actor X from camera position b is called a *parallel* camera placement. Filming X from position c yields an *internal* reverse placement. Shooting from position d results in an *apex* shot that shows both actors. Finally, filming from g is called an *external* reverse placement.

Cinematographers have identified that certain "cutting heights" make for pleasing compositions while others yield ugly results (*e.g.*, an image of a man cut off at the ankles). There is a set of (roughly) five useful camera distances (Arijon 1976, p. 18). An *extreme closeup* cuts at the neck; a *closeup* cuts under the chest or at the waist; a *medium view* cuts at the crotch or under the knees; a *full view* shows the entire person; and a *long view* provides a distant perspective.

Heuristics and Constraints

Filmmakers have articulated numerous heuristics for selecting good shots and have informally specified constraints to be placed on successive shots to lead to good scenes. Several of the more important rules include (Arijon 1976):

- *Parallel editing:* Story lines (visualized as scenes) should alternate between different characters, locations, or times.
- *Only show peak moments of the story:* Repetitive moments from a narrative should be deleted.
- *Don't cross the line:* Once an initial shot is taken from the left or right side of the line, subsequent shots should maintain that side, unless a neutral, establishing shot is used to show the transition from one side to the other. This rule ensures that successive shots of a moving actor will maintain the direction of apparent motion.
- *Let the actor lead:* The actor should initiate all movement, with the camera following; conversely, the camera should come to rest a little before the actor.
- *Break movement:* A scene illustrating motion should be broken into at least two shots. Typically, each shot is cut so that the actor appears to move across half the screen area. A change of the camera-to-subject distance should also be made in the switch.

Idioms

Perhaps the most significant invention of cinematographers is the notion of an *idiom* — a stereotypical way to capture some specific action as a series of shots. For example, in a dialogue between two people, a filmmaker might begin with an apex view of both actors, and then alternate views of each, at times using internal reverse placements and at times using external reverse. While there is an infinite variety of idioms, film directors have learned to rely on a small subset of these. Indeed, film books (*e.g.*, (Arijon 1976)) are primarily a compilation of idioms along with a discussion of the situations when a filmmaker should prefer one idiom over another. Figure 2 presents a three-shot idiom that serves as an extended example throughout the remainder of this paper. The idiom, adapted from Figure 13.2 of Arijon's text (1976), provides a method for depicting short-range motion of one actor approaching another. The first shot is a closeup; actor X begins in the center of the screen and exits left. The second shot begins with a long view of actor Y; actor X enters from off-screen right, and the shot ends when X reaches the center. The final shot begins with a medium view of Y, with actor X entering from off-screen right and stopping at center.

DCCL

This section provides an informal description of the Declarative Camera Control Language DCCL. The specification of DCCL is important because it allows CPS to formalize, encode, and implement common film idioms, such as the one presented in Figure 2.

[2] A notable exception is Alfred Hitchcock's *Rope*, which was filmed in a single shot, albeit with disguised breaks.

```
(AcFilmIdiom name Arijon-13-2
  :parameter (AcParamApproach :actor1 :actor2 :start :stop)
  :line (AcLineIdiom :primary ?actor1 :other ?actor2 :side left)
  (AcFilmShot name shot1
    (AcFragGoBy name frag1
      :time ?start :primary-moment beginning :entry-pos center :exit-pos out-left
      :placement (AcPlaceInternal :primary ?actor1 :other ?actor2 :range closeup :primary-side center)))
  (AcFilmShot name shot2
    (AcFragGoBy name frag2
      :time ?frag3.first-tick :primary-moment end :entry-pos on-right :exit-pos center
      :placement (AcPlaceExternal :near ?actor1 :far ?actor2
                                  :primary-subject near :range longshot :primary-side center)))
  (AcFilmShot name shot3
    (AcFragGoBy name frag3
      :time ?stop :primary-moment end :entry-pos out-right :exit-pos right12
      :placement (AcPlaceApex :primary ?actor1 :other ?actor2 :range mediumshot :primary-side right12))))
```

Figure 3: DCCL code corresponding to the idiom depicted in Figure 2.

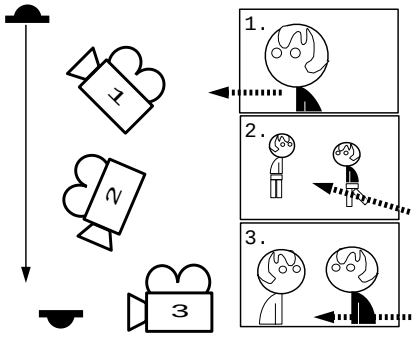

Figure 2: (Adapted from figure 13.2 of (Arijon 1976)). A common idiom for depicting short range movement of one actor approaching another. Camera positions are shown on the left side of the figure; the resulting image is shown on the right. Arrows indicate motion of actors into and out of the screen.

There are four basic primitive concepts in DCCL: *fragments*, *views*, *placements*, and *movement endpoints*; these primitives are combined to specify higher-level constructs such as shots and idioms.

Fragments

In the previous section, we discussed how cinematographers treat a shot as the primitive building block in a film. For our automated system, we have found it useful to further decompose each shot into a collection of one or more *fragments*. A fragment specifies an interval of time during which the camera is in a static position and orientation or is performing a single simple motion. DCCL defines five fragment types (illustrated schematically in Figure 4).

A fully-specified fragment requires additional information. Some of these arguments are obvious — for example, to track the motion of an actor, one must specify which actor to track and over what time interval to roll the film. In addition, one must also specify the desired camera *range* — extreme, closeup, medium, full, or long, as well as the *placement* of the camera relative to the actor (or actors) — internal, external, parallel, and apex.

Three of the fragments, go-by, panning, and tracking, require an additional argument, called the *primary moment*, which specifies the moment at which the placement command is to take effect during a shot in which there is motion.

Finally, two of these fragments, go-by and panning, require another argument called a *movement endpoint*, which is used to indicate the range of motion to be covered by the actor relative to the screen.[3] As Figure 5 illustrates, DCCL recognizes seven movement-endpoint keywords.

Note that although the movement-endpoint keywords refer to locations on the screen, they are used to calculate the *temporal* duration of go-by and panning fragments.[4] For example, the first shot of the idiom of Figure 2 can be defined as a go-by moving from center to out-left.

Shots and Idioms

In many cases, a shot is composed of a single fragment, but sometimes it is effective to merge several fragments together to form a more complex shot. For example, one can start with a panning fragment in

[3] To understand why this argument is necessary, recall that a go-by fragment results in a static camera position directed at an actor who moves across the field of view. An example of a go-by fragment is the first shot of Figure 2. Note that Arijon (1976) expresses the shot not by specifying the temporal duration, but rather by indicating (with arrows) the range of motion that the actor should cover while the film is rolling. DCCL uses movement endpoints to allow the same type of declarative specification and relies upon the compiler to calculate the temporal bounds that will yield the proper range of motion.

[4] This explains the definition of out-right and out-left. Arijon (1976) specifies that shots in which an actor moves off-screen (or onto the screen) should be cut three frames after (of before) the actor disappears (or appears). As a result we define out-right and out-left in terms of the distance traveled while three frames transpire.

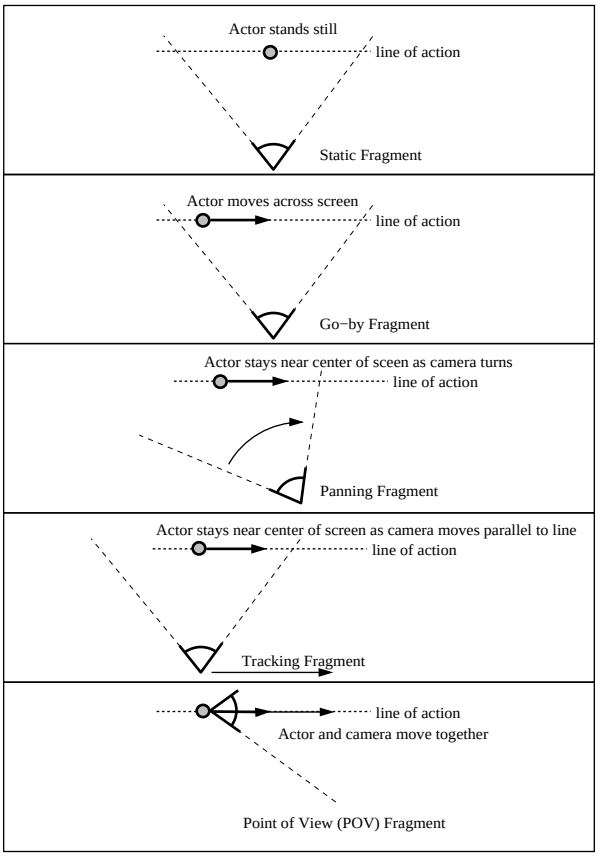

Figure 4: DCCL fragments specify the type of camera motion.

which a running actor moves from out-left into the center of the screen, then shift to a tracking shot by terminating camera rotation and increasing its lateral motion to match that of the actor (Figure 6). Multifragment shots typically combine panning, tracking and go-by fragments in different orders.

In multifragment shots, it is often important to be able to synchronize (in "simulation time") the end of one fragment with the beginning of the next. For this reason, DCCL supports the ability to export computed variables, such as the starting or ending time of a fragment, for use by other fragments in the same idiom. The duration of a scene can decrease if fragments do not cover the entire scene, producing "time contraction."

To define an idiom, one must specify the activities for which the idiom is deemed appropriate, a name, arguments (*e.g.*, actors and times), and a list of shot descriptions. For example, Figure 3 shows the actual DCCL encoding of the idiom illustrated in Figure 2; this idiom is a good choice for showing one actor approaching another.

The Camera Planning System

The Camera Planning System (CPS) codifies and implements the previously described cinematographic

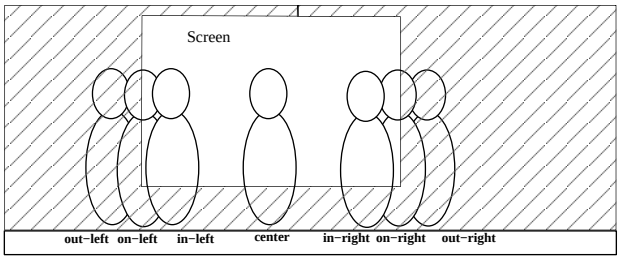

Figure 5: DCCL allows the user to delimit the temporal duration of go-by and panning fragments by specifying the desired initial and terminal locations of the actor on the screen.

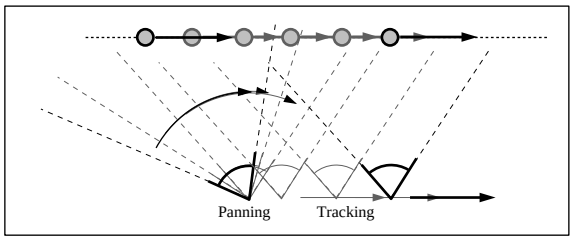

Figure 6: Schematic illustration of a shot composed of two fragments: a panning fragment that melds imperceptibly into a tracking fragment.

principles of camera placement for the case of simple movement sequences on the part of one or two actors. As input, CPS requires an animation trace: information about the positions and activities of each character, as well as directives stating which actors are to be filmed over what intervals of time. In our interactive game, this information is produced by a simple computer simulation generated in response to a user command. Given this trace information, the CPS chooses which subintervals of time to capture with the camera and from which vantage points and with which lenses (*i.e.*, what field of view). The animation can then be played back to the user using the intervals and camera placements selected by the CPS.

The primary data structure used by CPS is the *film tree* (Figure 7), which represents the film being generated. Of primary consequence are the scene and candidate idiom levels of the tree: each scene in the film is associated with one or more possible idioms, with each idiom representing a particular method of filming the parent scene. The CPS operates by expanding the film tree until it has compiled detailed solutions for each idiom, and then selecting appropriate candidate idioms and frames for each scene.

Internally, the CPS is implemented as a three-stage pipeline involving a *sequence planner*, DCCL *compiler*, and a *heuristic evaluator*, as shown in Figure 8. An *idiom database* (not shown) provides idiom specifications relevant to each scene in the animation being filmed.

The Sequence Planner

The current implementation of the CPS sequence planner is quite simple. Unlike the other portions of the CPS

Figure 8: The CPS is implemented as a three-stage pipeline.

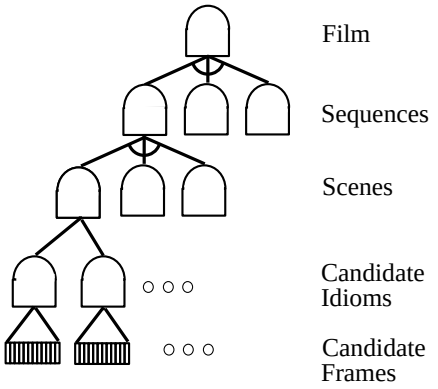

Figure 7: Successive modules in the CPS pipeline incrementally expand the film tree data structure.

Activity	*Solitary*	*Idioms*
Stopping/Starting	Y	1
X-Approaches-Y	N	2
X-Retreats-From-Y	N	2
X-Follows-Y	N	2
Moving	Y	2
Turning	Y	1
HeadTurning	Y	1
Stationary	Y	1
Looking	Y	1
Noticing	N	1
Picking Up	N	1
Holding	N	1

Table 1: Activity classifications for prototype game.

the code implementing the sequence planner is specific to the domain (plot) of the application (*e.g.*chase and capture interactions). As input the planner receives an animation trace describing the time-varying behavior of a group of actors. As output, the sequence planner produces a film tree (Figure 7) that is specified down to the scene level.

The animation trace specifies position, velocity, and joint positions for each actor for each frame of the animation. The trace also labels the activity being performed by each actor in each frame, as well as higher-level information encoded as a set of film sequences, with each film sequence including an interval of time, an actor to use as the *protagonist*, and (optionally) a second actor. In the current application, multiple film sequences are used to create parallel editing effects by having the CPS intermix scenes featuring one set of actors with scenes featuring a different set of actors (see accompanying videotape).

Given the information in the animation trace, the sequence planner generates scenes by first partitioning each film sequence according to the activities performed by the protagonist during the given sequence. For the current application we have identified ten activity types (Table 1). After partitioning a sequence, the sequence planner generates scenes parameterized by the activity, actors, and time interval of each partition.

Once the sequence planner has created the scene nodes, the CPS must instantiate the idiom schemata relevant for each scene. Relevance is determined by matching the scene activity against a list of applicable activities defined for each idiom. The current implementation of the database contains 16 idioms (Table 1). The planner instantiates idioms by substituting actual parameters (actor names, and scene start and ending times) for the placeholders specified in the idiom definitions. References to actor placements on the right or left sides will be automatically mirrored later, during the idiom solving process.

The DCCL Compiler

The DCCL compiler uses information about the movement of the actors to expand the fragments in each candidate idiom into an array of frame specifications, which can be played directly. Since a frame is fully constrained by the combination of its camera's 3D position, orientation, and field of view, the compiler need only generate an array of these values for each fragment in each shot in each candidate idiom.

In its simplest form, an idiom consists of a single shot that is composed of a single fragment, so we cover that case first. If the fragment has type pov, then compilation is trivial, so we assume that the fragment has type static, tracking, go-by, or panning. We decompose the compiler's job into four tasks:

1. Determine the appropriate primary moment(s).
2. Determine the set of possible frame specifications for each primary moment.
3. Calculate the temporal duration (length of the frame array) of the fragment given an initial frame specification.
4. Generate the interpolated specification of frame n from that of frame $n-1$.

Once these tasks have been completed, the compiler simply has to generate a frame array for each primary moment and frame specification. In the current version of CPS there are typically only two frame arrays corresponding to placing the camera on one side of the line of interest or the other. The task of choosing the

appropriate side is left up to the heuristic evaluator.[5]

The primary moment of a fragment defines the point in time at which the camera placement should conform to the placement specified for the fragment, and varies with the type of fragment. The go-by, tracking, and panning fragments specify the primary moment as either the first or last tick of the fragment. The static fragments, on the other hand, do not specify a primary moment, so in the current version of CPS we solve the placement for the first, last, and midpoint ticks of the fragment's time interval; the heuristic evaluator will later determine which solution looks best and prune the rest.

The fragment's placement (*e.g.*, internal, external, parallel, or apex), as specified in the idiom, combined with the location of the actors (from the animation trace) constrains the camera's initial position to lie on one of two vectors, according to the side of the line being used (Figure 1). The actual location on this vector is determined by the desired distance between the camera and the primary subject. This distance, in turn, is specified by the fragment's range (extreme, closeup, medium, full, or long) and the lens focal length. The compiler attempts to generate a set of appropriate placements using a normal lens (*e.g.*, a 60-degree field of view). The vector algebra behind these calculations is explained in (Blinn 1988).

The temporal duration of static, tracking, and pov fragments is specified explicitly as part of the DCCL specification. However, the duration of go-by and panning fragments must be computed from the movement-endpoint specification in conjunction with the actor's velocity.

The function used to update the camera position and orientation from one frame to the next depends on the type of fragment involved and the change in the actors' positions. Static and go-by fragments do not change camera position or orientation. The camera in tracking fragments maintains its orientation, but changes its position based on the actor's velocity vector. The camera in panning fragments maintains its position, but changes its orientation with angular velocity constrained by actor velocity and the distance at the closest approach to the camera (as determined by the primary moment).

Note that unlike the sequence planner, the DCCL compiler is completely domain-independent in that the compiler depends only on geometry and not on the plot or subject of the animation. Furthermore, the DCCL specifications in the idiom database are applicable across various animations; for example, the idioms in our database should apply to any animation with two-character interactions.

[5]Typically, the camera is restricted to one side of the line of interest. However, opportunities to "switch sides" sometimes present themselves, such as when an actor turns to walk in a neutral direction.

The Heuristic Evaluator

Since the film tree is an AND/OR graph, it represents many possible films.[6] The heuristic evaluator chooses the candidate idiom that renders a scene best, assigning each idiom a real-valued score by first scoring and pruning the frame arrays of its fragments. Note that it is not possible to estimate the quality of a fragment or idiom before it is compiled, because visual quality is determined by the complex, geometric interactions between the fragment type and the motions of the actors. A given idiom might work extremely well when the actors stand in one position relative to one another, yet work poorly given a different configuration.

The scoring mechanism is primarily focused towards evaluating inter-fragment criteria, namely:

- maintaining smooth transitions between camera placements;
- eliminating fragments which cause the camera to cross the line.

In addition, the scoring mechanism deals with certain intra-fragment behaviors that sometimes arise from the compilation phase of the CPS such as:

- penalizing very short or very long fragments;
- eliminating fragments in which the camera pans backwards.

After the evaluator has selected the best idiom for each scene to be included in the film, the camera planning process is complete. CPS concatenates the frame arrays for all idiom nodes remaining in the film tree and outputs the corresponding sequence of frames to the player for rendering.

Note that while the evaluator's rules are heuristic, they are also domain-independent within the domain of film and animation: each rule encodes broadly-applicable knowledge about which types of shots look good on film, and which do not.

Sample Application

We are particularly interested in interactive uses of automatic cinematography. Therefore, we decided to build a simple interactive game that would use CPS to film actions commanded by a human player. The basic plot of the game is very simple. The main character, Bob, must walk around a virtual world (in our case, SGI's Performer Town) looking for the Holy Grail. The game is made more interesting by the introduction of Fido, an evil dog that seeks to steal the Grail. From time to time, the game generates animations of Fido's activities and instructs CPS to edit these animations into the action. Eventually, Bob and Fido interact: if Fido gets the grail, Bob has to chase Fido in order to get it back. The user commands Bob through a pop-up menu. These commands fall into four basic categories:

[6]Indeed, if there are n scenes and each scene has k candidate idioms as children, then the film tree represents n^k possible idiom combinations.

telling Bob to look in a particular direction, to move to a particular point on the screen, to pick up an object, or to chase another actor.

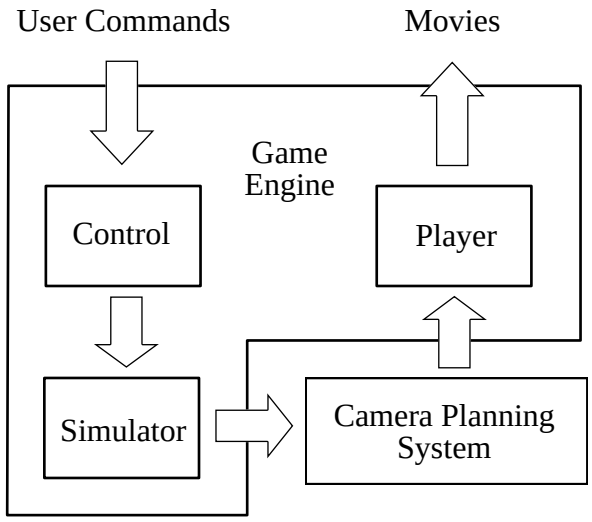

Figure 9: Overall context of CPS

The implementation of the game and its various animation/simulation modules was done in C++ using the IRIS Performer toolkit running on SGI workstations, and based partially on the perfly application provided by SGI (a Performer-based walkthrough application). The game operates as a finite-state machine that produces animation traces as the user issues commands to the game engine, with the CPS acting as a separate library whose sole inputs are the animation trace and the database of idioms (Figure 9). The game itself (not counting CPS or the code present in the existing perfly application) required approximately 10,000 lines of C++ code. The CPS system is also written in C++ (despite the Lisp-like appearance of DCCL) and implemented with about 19,000 lines of code.

The sample game interaction presented at the end of our video is intended to demonstrate a number of the activities possible in the game, as well as the various DCCL idioms. For presentation purposes, the planning time required by CPS was edited out of the video; Table 2 gives performance data taken from a similar runthrough of the game on an SGI Onyx.

Related Work

The subject of using principles from cinematography to control camera positions and scene structure has received relatively little attention in the computer graphics or AI communities. We survey most of the related work here.

He, Cohen, and Salesin (1996) have developed a system for controlling camera placement in real-time using some of the ideas behind DCCL. Their work focuses on filming dialogues between multiple animated characters, and uses a finite state machine to select and generate camera positions.

A number of systems have been described for automatically placing the camera in an advantageous position when performing a given interactive task (Gleicher & Witkin 1992; Mackinlay, Card, & Robertson 1990; Phillips, Badler, & Granieri 1992). However, these systems neither attempt to create sequences of scenes, nor do they apply rules of cinematography in developing their specifications.

In work that is closer to our own, Karp and Feiner (Karp & Feiner 1990; 1993) describe an animation planning system for generating automatic presentations. Their emphasis is on the planning engine itself, whereas the work described in this paper is more concerned with the problem of defining a high-level declarative language for encoding cinematic expertise. Thus, the two approaches complement each other.

Strassman (Strassman 1994) reports on Divaldo, an ambitious experiment to create a prototype system for "desktop theatre." Unlike our focus on camera placement, Strassman attempts to create semi-autonomous actors who respond to natural language commands. CPS is also complementary to Divaldo.

Drucker *et al.* (Drucker, Galyean, & Zeltzer 1992; Drucker & Zeltzer 1994; 1995) are concerned with the problem of setting up the optimal camera position for individual shots, subject to constraints. Specific camera parameters are automatically tuned for a given shot based on a general-purpose continuous optimization paradigm. In our work, a set of possible cameras is fully specified by the shot descriptions in DCCL and the geometry of the scene. The final selection from among this set of different shots is made according to how well each shot covers the scene. Our approach avoids the need for generic optimization searches, and it is guaranteed to result in a common shot form. The cost is a greatly reduced set of possible camera specifications.

Several useful texts derive low-level camera parameters, given the geometry of the scene (Foley *et al.* 1990; Hearn & Baker 1994; Blinn 1988).

Conclusion

We close by summarizing the contributions of this paper and describing the directions we intend to pursue in future work. The main contributions of this paper include:

- Surveying established principles from filmmaking that can be used in a variety of computer graphics applications.
- Describing a high-level language, DCCL, for specifying camera shots in terms of the desired positions and movements of actors across the screen. We have argued that DCCL represents cinematic knowledge at the same abstraction level as expert directors and producers by encoding sixteen idioms from a film textbook (Arijon 1976) (*e.g.*, Figure 2).

Command	*Num. Frames*	*Scenes Generated*	*CPU Time (s)*
Pick Up Grail	733	9	47.63
Pick Up Net	451	4	44.74
Catch Dog	603	4	32.74
Walk (long range)	701	4	11.52
Walk (med range)	208	4	6.74
Look Right	87	3	5.6
Walk (short range)	158	4	5.12

Table 2: Typical CPS Performance

- Presenting a domain-independent compiler that solves DCCL constraints and dynamically controls the camera.
- Describing a domain-independent heuristic evaluator that ranks the quality of a shot specification using detailed geometric information and knowledge of desirable focal lengths, shot durations, etc.
- Describing a fully-implemented film camera planning system (CPS) that uses the DCCL compiler and heuristic evaluator to synthesize short animated scenes from 3D data produced by an independent, interactive application.
- Incorporating CPS into a prototype game, and demonstrating sample interactions in the game (see videotape).

Acknowledgements

This research was funded in part by Office of Naval Research grants N00014-94-1-0060 and N00014-95-1-0728, National Science Foundation grants IRI-9303461 and CCR-9553199, ARPA/Rome Labs grant F30602-95-1-0024, an Alfred P. Sloan Research Fellowship (BR-3495), and industrial gifts from Interval, Microsoft, Rockwell, and Xerox.

References

Arijon, D. 1976. *Grammar of the Film Language*. New York: Communication Arts Books, Hastings House, Publishers.

Blinn, J. 1988. Where am I? What am I looking at? *IEEE Computer Graphics and Applications* 76–81.

Christianson, D. B., Anderson, S. E., He, L., Weld, D. S., Salesin, D. H., and Cohen, M. F. 1996. Declarative camera control for automatic cinematography. TR UW-CSE-96-02-01, University of Washington Department of Computer Science and Engineering.

Drucker, S. M., and Zeltzer, D. 1994. Intelligent camera control in a virtual environment. In *Proceedings of Graphics Interface '94*, 190–199. Banff, Alberta, Canada: Canadian Information Processing Society.

Drucker, S. M., and Zeltzer, D. 1995. Camdroid: A system for intelligent camera control. In *Proceedings of the SIGGRAPH Symposium on Interactive 3D Graphics '95*.

Drucker, S. M., Galyean, T. A., and Zeltzer, D. 1992. CINEMA: A system for procedural camera movements. In Zeltzer, D., ed., *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, 67–70.

Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. 1990. *Computer Graphics, Principles and Practice*. Reading, Massachusetts: Addison-Wesley Publishing Company, second edition.

Gleicher, M., and Witkin, A. 1992. Through-the-lens camera control. In Catmull, E. E., ed., *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, 331–340.

He, L., Cohen, M. F., and Salesin, D. H. 1996. Virtual cinematography: A paradigm for automatic real-time camera control and directing. To appear at SIGGRAPH '96.

Hearn, D., and Baker, M. P. 1994. *Computer Graphics*. Englewood Cliffs, New Jersey: Prentice Hall, second edition.

Karp, P., and Feiner, S. 1990. Issues in the automated generation of animated presentations. In *Proceedings of Graphics Interface '90*, 39–48.

Karp, P., and Feiner, S. 1993. Automated presentation planning of animation using task decomposition with heuristic reasoning. In *Proceedings of Graphics Interface '93*, 118–127. Toronto, Ontario, Canada: Canadian Information Processing Society.

Lukas, C. 1985. *Directing for Film and Television*. Garden City, N.Y.: Anchor Press/Doubleday.

Mackinlay, J. D., Card, S. K., and Robertson, G. G. 1990. Rapid controlled movement through a virtual 3D workspace. In Baskett, F., ed., *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, 171–176.

Mascelli, J. V. 1965. *The Five C's of Cinematography*. Hollywood: Cine/Grafic Publications.

Phillips, C. B., Badler, N. I., and Granieri, J. 1992. Automatic viewing control for 3D direct manipulation. In Zeltzer, D., ed., *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, 71–74.

Strassman, S. 1994. Semi-autonomous animated actors. In *Proceedings of the AAAI-94*, 128–134.