CamDroid: A System for Implementing Intelligent Camera Control

Steven M. Drucker
MIT Media Lab

David Zeltzer
MIT Research Laboratory for Electronics

Massachusetts Institute of Technology
Cambridge, MA. 02139, USA
smd@media.mit.edu
dz@vetrec.mit.edu

Abstract

In this paper, a method of encapsulating camera tasks into well defined units called "camera modules" is described. Through this encapsulation, camera modules can be programmed and sequenced, and thus can be used as the underlying framework for controlling the virtual camera in widely disparate types of graphical environments. Two examples of the camera framework are shown: an agent which can film a conversation between two virtual actors and a visual programming language for filming a virtual football game.

Keywords: Virtual Environments, Camera Control, Task Level Interfaces.

1. Introduction

Manipulating the viewpoint, or a synthetic camera, is fundamental to any interface which must deal with a three dimensional graphical environment, and a number of articles have discussed various aspects of the camera control problem in detail [3, 4, 5, 19]. Much of this work, however, has focused on techniques for directly manipulating the camera.

In our view, this is the source of much of the difficulty. Direct control of the six degrees of freedom (DOFs) of the camera (or more, if field of view is included) is often problematic and forces the human VE participant to attend to the interface and its "control knobs" in addition to — or instead of — the goals and constraints of the task at hand. In order to achieve *task level* interaction with a computer-mediated graphical environment, these low-level, direct controls, must be abstracted into higher level camera primitives, and in turn, combined into even higher level interfaces. By clearly specifying what specific tasks need to be accomplished at a particular unit of time, a wide variety of interfaces can be easily constructed. This technique has already been successfully applied to interactions within a Virtual Museum [8].

1995 Symposium on Interactive 3D Graphics, Monterey CA USA
© 1995 ACM 0-89791-736-7/95/0004...$3.50

2. Related Work

Ware and Osborne [19] described several different metaphors for exploring 3D environments including "scene in hand," "eyeball in hand," and "flying vehicle control" metaphors. All of these use a 6 DOF input device to control the camera position in the virtual environment. They discovered that flying vehicle control was more useful when dealing with enclosed spaces, and the "scene in hand" metaphor was useful in looking at a single object. Any of these metaphors can be easily implemented in our system.

Mackinlay et al [16] describe techniques for scaling camera motion when moving through virtual spaces, so that, for example, users can always maintain precise control of the camera when approaching objects of interest. Again, it is possible to implement these techniques using our camera modules.

Brooks [3,4] discusses several methods for using instrumented mechanical devices such as stationary bicycles and treadmills to enable human VE participants to move through virtual worlds using natural body motions and gestures. Work at Chapel Hill, has, of course, focused for some time on the architectural "walk-through," and one can argue that such direct manipulation devices make good sense for this application. While the same may be said for the virtual museum, it is easy to think of circumstances — such as reviewing a list of paintings — in which it is not appropriate to require the human participant to physically walk or ride a bicycle. At times, one may wish to interact with topological or temporal abstractions, rather than the spatial. Nevertheless, our camera modules will accept data from arbitrary input devices as appropriate.

Blinn [2] suggested several modes of camera specification based on a description of what should be placed in the frame rather than just describing where the camera should be and where it should be aimed.

Phillips et al suggest some methods for automatic viewing control [18]. They primarily use the "camera in hand" metaphor for viewing human figures in the Jack™ system, and provide automatic features for maintaining smooth visual transitions and avoiding viewing obstructions. They do not deal with the problems of navigation, exploration or presentation.

Karp and Feiner describe a system for generating automatic presentations, but they do not consider interactive control of the camera [12].

Gleicher and Witkin [10] describe a system for controlling the movement of a camera based on the screen-space projection of an object, but their system works primarily for manipulation tasks.

Our own prior work attempted to establish a procedural framework for controlling cameras [7]. Problems in constructing generalizable procedures led to the current, constraint-based framework described here. Although this paper does not concentrate on methods for satisfying multiple constraints on the camera position, this is an important part of the overall camera framework we outline here. For a more complete reference, see [9]. An earlier form of the current system was applied to the domain of a Virtual Museum [8].

3. CamDroid System Design

This framework is a formal specification for many different types of camera control. The central notion of this framework is that camera placement and movement is usually done for particular reasons, and that those reasons can be expressed formally as a number of primitives or constraints on the camera parameters. We can identity these constraints based on analyses of the tasks required in the specific job at hand. By analyzing a wide enough variety of tasks, a large base of primitives can be easily drawn upon to be incorporated into a particular task-specific interface.

3.1 Camera Modules

A camera module represents an encapsulation of the constraints and a transformation of specific user controls over the duration that a specific module is active. A complete network of camera modules with branching conditions between modules incorporates user control, constraints, and response to changing conditions in the environment over time.

Our concept of a camera module is similar to the concept of a *shot* in cinematography. A shot represents the portion of time between the starting and stopping of filming a particular scene. Therefore a shot represents continuity of all the camera parameters over that period of time. The unit of a single camera module requires an additional level of continuity, that of continuity of *control* of the camera. This requirement is added because of the ability in computer graphics to identically match the camera parameters on either side of a cut, blurring the distinction of what makes up two separate shots. Imagine that the camera is initially pointing at character A and following him as he moves around the environment. The camera then pans to character B and follows her for a period of time. Finally the camera pans back to character A. In cinematic terms, this would be a single shot since there was continuity in the camera parameters over the entire period. In our terms, this would be broken down into four separate modules. The first module's task is to follow character A. The second module's task would be to pan from A to B in a specified amount of time. The third module's task would be to follow B. And finally the last module's task would be to pan back from B to A. The notion of breaking this cinematic shot into 4 modules does not specify implementation, but rather a formal description of the goals or constraints on the camera for each period of time.

As shown in figure 1, the generic module contains the following components:

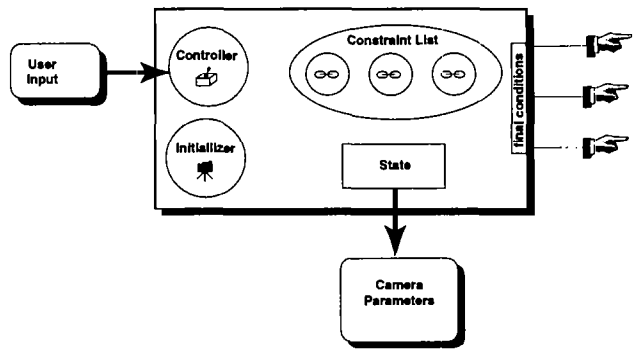

Figure 1: Generic camera module containing a controller, an initializer, a constraint list, and local state

- the local state vector. This must always contain the camera position, camera view normal, camera "up" vector, and field of view. State can also contain values for the camera parameter derivatives, a value for time, or other local information specific to the operation of that module. While the module is active, the state's camera parameters are output to the renderer.
- initializer. This is a routine that is run upon activation of a module. Typical initial conditions are to set up the camera state based on a previous module's state.
- controller. This component translates user inputs either directly into the camera state or into constraints. There can be at most one controller per module.
- constraints to be satisfied during the time period that the module is active. Some examples of constraints are as follows:
  - maintain the camera's up vector to align with world up.
  - maintain height relative to the ground
  - maintain the camera's gaze (i.e. view normal) toward a specified object
  - make sure a certain object appears on the screen.
  - make sure that several objects appear on the screen
  - zoom in as much as possible

In this system, the constraint list can be viewed simply as a black box that produces values for some DOFs of the camera. The constraint solver combines these constraints using a constrained optimizing solver to come up with the final camera parameters for a particular module. The camera optimizer is discussed extensively in [9]. Some constraints directly produce values for a degree of freedom, for example, specifying the up vector for the camera or the height of the camera. Some involve calculations that might produce multiple DOFs, such as adjusting the view normal of the camera to look at a particular object. Some, like a path planning constraint discussed in [8] are quite complicated, and generate a series of DOFs over time through the environment based on an initial and final position.

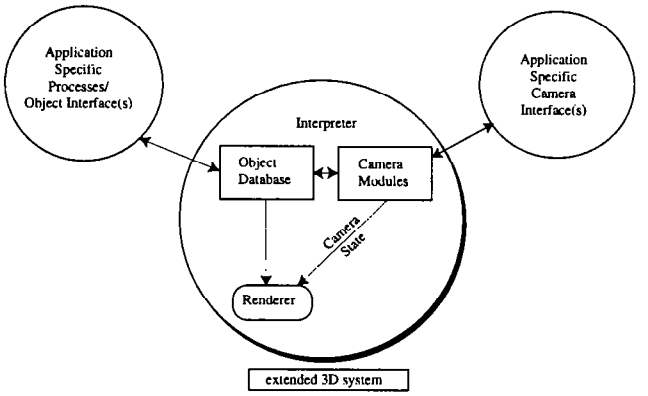

Figure 2: Overall CamDroid System

3.2 The CamDroid System

The overall system for the examples given in this paper is shown in figure 2.

The CamDroid System is an extension to the 3D virtual environment software testbed developed at MIT [6]. The system is structured this way to emphasize the division between the virtual environment database, the camera framework, and the interface that provides access to both. The CamDroid system contains the following elements.

- A general interpreter that can run pre-specified scripts or manage user input. The interpreter is an important part in developing the entire runtime system. Currently the interpreter used is TCL with the interface widgets created with TK [17]. Many commands have been embedded in the system including the ability to do dynamic simulation, visibility calculations, finite element simulation, matrix computations, and various database inquiries. By using an embedded interpreter we can do rapid prototyping of a virtual environment without sacrificing too much performance since a great deal of the system can still be written in a low level language like C. The addition of TK provides convenient creation of interface widgets and interprocess communication. This is especially important because some processes might need to perform computation intensive parts of the algorithms; they can be offloaded onto separate machines.
- A built-in renderer. This subsystem can use either the hardware of a graphics workstation (currently SGIs and HPs are supported), or software to create a high quality antialiased image.
- An object database for a particular environment.
- Camera modules. Described in the previous section. Essentially, they encapsulate the behavior of the camera for different styles of interaction. They are prespecified by the user and associated with various interface widgets. Several widgets can be connected to several camera modules. The currently active camera module handles all user inputs and attempts to satisfy all the constraints contained within the module, in order to compute camera parameters which will be passed to the renderer when creating the final image. Currently, only one camera module is active at any one time, though if there were multiple viewports, each of them could be assigned a unique camera.

4. Example: Filming a conversation

The interface for the conversation filming example is based on the construction of a software agent which perceives changes in limited aspects of the environments and uses a number of primitives to implement agent behaviors. The sensors detect movements of objects within the environment and can perceive which character is designated to be talking at any moment.

In general, the position of the camera should be based on conventional techniques that have been established in filming a conversation. Several books have dissected conversations and come up with simplified rules for an effective presentation [1, 14]. The conversation filmer encapsulates these rules into camera modules which the software agent calls upon to construct (or assist a director in the construction of) a film sequence.

4.1 Implementation

The placement of the camera is based on the position of the two people having the conversation (see figure 3). However, more important than placing the camera in the approximate geometric relationship shown in figure 3 is the positioning of the camera based on what is being framed within the image.

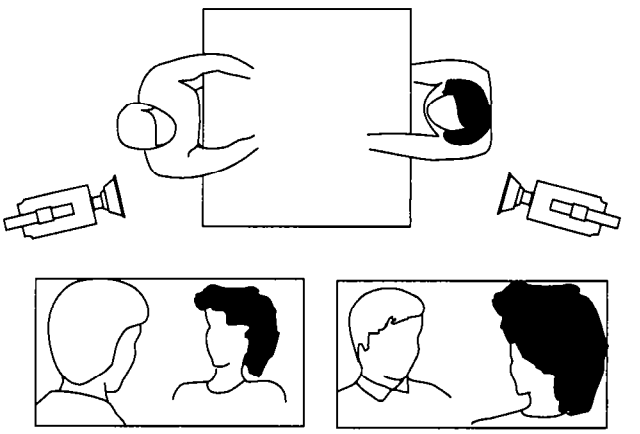

Figure 3: Filming a conversation [Katz88].

Constraints for an over-the-shoulder shot:

- The height of the character facing the view should be approximately 1/2 the size of the frame.
- The person facing the view should be at about the 2/3 line on the screen.
- The person facing away should be at about the 1/3 line on the screen.
- The camera should be aligned with the world up.
- The field of view should be between 20 and 60 degrees.
- The camera view should be as close to facing directly on to the character facing the viewer as possible.

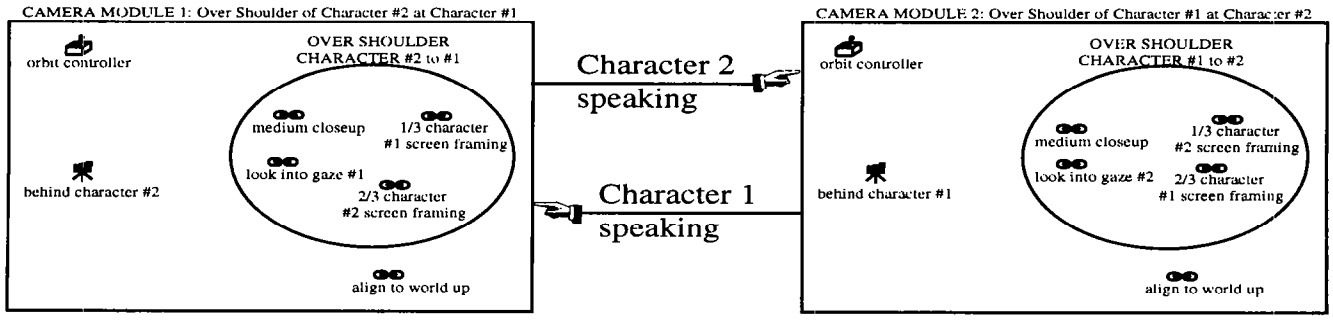

Figure 4: Two interconnected camera modules for filming a conversation

Constraints for a corresponding over-the-shoulder shot:

- The same constraints as described above but the people should not switch sides of the screen; therefore the person facing towards the screen should be placed at the 1/3 line and the person facing away should be placed at the 2/3 line.

Figure 3 can be used to find the initial positions of the cameras if necessary, but the constraint solver contained within each camera module makes sure that the composition of the screen is as desired.

Figure 4 shows how two camera modules can be connected to automatically film a conversation.

A more complicated combination of camea modules can be incorporated as the behaviors of a simple software agent. The agent contains a rudimentary reactive planner which pairs camera behaviors (combination of camera primitives) in response to sensed data. The agent has two primary sets of camera behaviors: one for when character 1 is speaking; and one for when character 2 is speaking. The agent needs to have sensors which can "detect" who is speaking and direct a camera module from the desired set of behaviors to become active. Since the modules necessarily keep track of the positions of the characters in the environment, the simulated actors can move about while the proper screen composition is maintained.

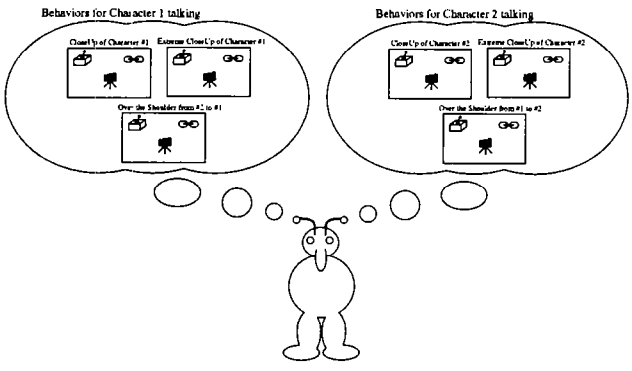

Figure 5: Conversation filming agent and its behaviors.

Figure 6 shows an over-the-shoulder shot automatically generated by the conversation filming agent.

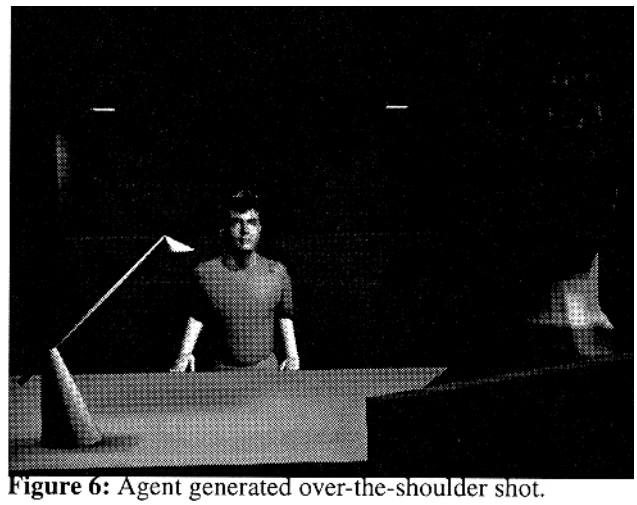

Figure 6: Agent generated over-the-shoulder shot.

5. Example: the Virtual Football Game

The virtual football game was chosen as an example because there already exists a methodology for filming football games that can be called upon as a reference for comparing the controls and resultant output of the virtual football game. Also, the temporal flow of the football game is convenient since it contains starting and stopping points, specific kinds of choreographed movements, and easily identifiable participants. A visual programming language for combining camera primitives into camera behaviors was explored. Finally, an interface, on top of the visual programming language, based directly on the way that a conventional football game is filmed, was developed.

It is important to note that there are significant differences between the virtual football game and filming a real football game. Although attempts were made to make the virtual football game realistic— three-dimensional video images of players were incorporated and football plays were based on real plays [15] —this virtual football game is intended to be a testbed for intelligent camera control rather than a portrayal of a real football game.

5.1 Implementation

Figure 7 shows the visual programming environment for the camera modules. Similar in spirit to Haeberli's ConMan [11] or Kass's GO [13], the system allows the user to connect camera modules,

and drag and drop initial conditions and constraints, in order to control the output of the CamDroid system. The currently active camera module's camera state is used to render the view of the graphical environment. Modules can be connected together by drawing a line from one module to the next. A boolean expression can then be added to the connector to indicate when control should be shifted from one module to the connected module. It is possible to set up multiple branches from a single module. At each frame, the branching conditions are evaluated and control is passed to the first module whose branching condition evaluates to TRUE.

Constraints can be instanced from existing constraints, or new ones can be created and the constraint functions can be entered via a text editor. Information for individual constraints can be entered via the keyboard or mouse clicks on the screen. When constraints are dragged into a module, all the constraints in the module are included during optimization. Constraints may also be grouped so that slightly higher level behaviors composed of a group of low level primitives may be dragged directly into a camera module.

Initial conditions can be dragged into the modules to force the minimization to start from those conditions. Initial conditions can be retrieved at any time from the current state of the camera. Camera modules can also be indicated to use the current state to begin optimization when control is passed to them from other modules.

Controllers can also be instanced from a palette of existing controllers, or new ones created and their functions entered via a text editor. If a controller is dragged into the module, it will translate the actions of the user subject to the constraints within the module. For example, a controller that will orbit about an object may be added to a module which constrains the camera's up vector to align with the world up vector.

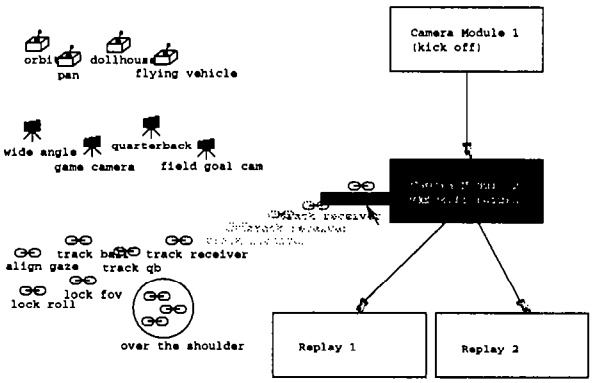

Figure 7: Visual Programming Environment for camera modules

The end-user does not necessarily wish to be concerned with the visual programming language for camera control. An interface that can be connected to the representation used for the visual programming language is shown in Figure 7. The interface provides a mechanism for setting the positions and movements of the players within the environment, as well as a way to control the virtual cameras. Players can be selected and new paths drawn for them at any time. The players will move along their paths in response to clicking on the appropriate buttons of the football play controller. Passes can be indicated by selecting the appropriate players at the appropriate time step and pressing the pass button on the play controller.

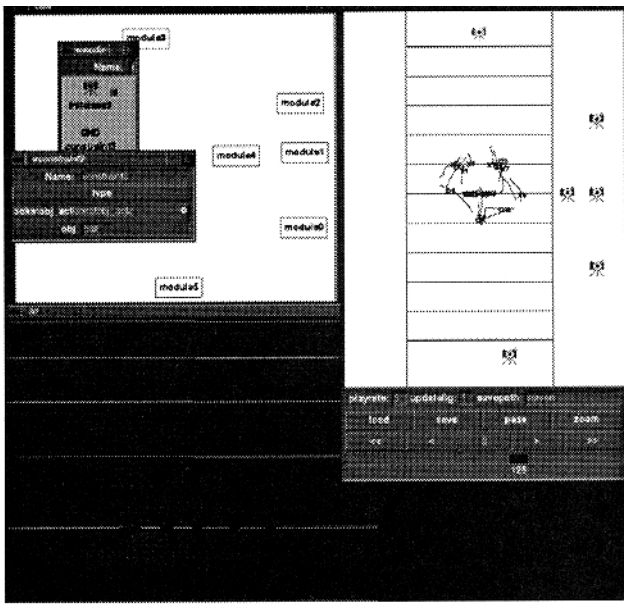

Figure 8: The virtual football game interface

The user can also select or move any of the camera icons and the viewpoint is immediately shifted to that of the camera. Essentially, pressing one of the camera icons activates a camera module that has already been set up with initial conditions and constraints for that camera. Cameras can be made to track individual characters or the ball by selecting the players with the middle mouse button. This automatically adds a tracking constraint to the currently active module. If multiple players are selected, then the camera attempts to keep both players within the frame at the same time by adding multiple tracking constraints. The image can currently be fine-tuned by adjusting the constraints within the visual programming environment. A more complete interface would provide more bridges between the actions of the user on the end-user interface and the visual programming language..

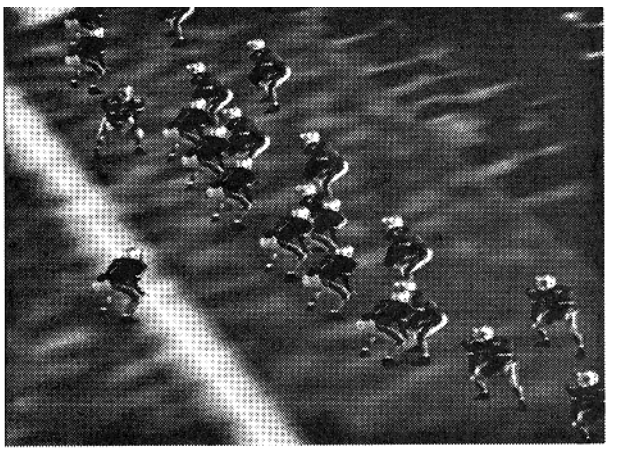

Figure 9: View from "game camera" of virtual football game.

6. Results

We have implemented a variety of applications from a disparate set of visual domains, including the virtual museum [8], a mission planner [21], and the conversation and football game described in this paper. While formal evaluations are notoriously difficult, we did enlist the help of domain experts who could each observe and comment on the applications we have implemented. For the conversation agent, our domain expert was MIT Prof.essor Glorianna Davenport, in her capacity as an accomplished documentary filmmaker. For the virtual football game, we consulted with Eric Eisendratt, a sports director for WBZ-TV, Boston. In addition, MIT Professor Tom Sheridan was an invaluable source of expertise on teleoperation and supervisory control. A thorough discussion of the applications, including comments of the domain experts, can be found in [9].

7. Summary

A method of encapsulating camera tasks into well defined units called "camera modules" has been described. Through this encapsulation, camera modules can be designed which can aid a user in a wide range of interaction with 3D graphical environments. The CamDroid system uses this encapsulation, along with constrained optimization techniques and visual programming to greatly ease the development of 3D interfaces. Two interfaces to distinctly different environments have been demonstrated in this paper.

8. Acknowledgements

This work was supported in part by ARPA/Rome Laboratories, NHK (Japan Broadcasting Co.), the Office of Naval Research, and equipment gifts from Apple Computer, Hewlett-Packard, and Silicon Graphics.

9. References

1. Arijon, D., *Grammar of the Film Language.* 1976, Los Angeles: Silman-James Press.

2. Blinn, J., Where am I? What am I looking at? *IEEE Computer Graphics and Applications,* July 1988.

3. Brooks, F.P., Jr. Grasping Reality Through Illusion -- Interactive Graphics Serving Science. *Proc. CHI '88.* May 15-19, 1988.

4. Brooks, F.P., Jr. Walkthrough -- A Dynamic Graphics System for Simulating Virtual Buildings. *Proc. 1986 ACM Workshop on Interactive 3D Graphics.* October 23-24, 1986.

5. Chapman, D. and C. Ware. Manipulating the Future: Predictor Based Feedback for Velocity Control in Virtual Environment Navigation . *Proc. 1992 Symposium on Interactive 3D Graphics.* 1992. Cambridge MA: ACM Press.

6. Chen, D. T. and D. Zeltzer. The 3d Virtual Environment and Dynamic Simulation System. Cambridge MA, Technical Memo. MIT Media Lab. August, 1992.

7 Drucker, S., T. Galyean, and D. Zeltzer. CINEMA: A System for Procedural Camera Movements. *Proc. 1992 Symposium on Interactive 3D Graphics.* 1992. Cambridge MA: ACM Press.

8. Drucker, S. M. and D. Zeltzer. Intelligent Camera Control for Virtual Environments. *Graphics Interface '94.* 1994.

9. Drucker, S.M *Intelligent Camera Control for Graphical Environments.* PhD. Thesis. MIT Media Lab. 1994.

10. Gleicher, M..A.W. Through-the-Lens Camera Control. *Computer Graphics.* 26(2): pp. 331-340. 1992

11. Haeberli, P.E., ConMan: A Visual Programming Language for Interactive Graphics. *Computer Graphics.* 22(4): pp. 103-111. 1988

12. Karp, P. and S.K. Feiner. Issues in the automated generation of animated presentations. *Graphics Interface '90.* 1990.

13. Kass, M. GO: A Graphical Optimizer. in ACM SIGGRAPH 91 Course Notes, Introduction to Physically Based Modeling. July 28-August 2, 1991. Las Vegas NM.

14. Katz, S.D., *Film Directing Shot by Shot: Visualising from Concept to Screen.* 1991, Studio City, CA: Michael Weise Productions.

15. Korch, R. *The Official Pro Football Hall of Fame.* New York, Simon & Schuster, Inc. 1990.

16. Mackinlay, J. S., S. Card, et al. Rapid Controlled Movement Through a Virtual 3d Workspace. *Computer Graphics* 24(4): 171-176. 1990.

17. Ousterhout, J. K. Tcl: An Embeddable Command Language. *Proc. 1990 Winter USENIX Conference.* 1990.

18. Philips, C.B.N.I.B., John Granieri. Automatic Viewing Control for 3D Direct Manipulation. *Proc. 1992 Symposium on Interactive 3D Graphics.* 1992. Cambridge, MA.: ACM Press.

19. Ware, C. and S. Osborn. Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. *Proc. 1990 Symposium on Interactive 3D Graphics,* Snowbird, Utah, 1990. ACM Press.

20. Zeltzer, D. Autonomy, Interaction and Presence. *Presence: Teleoperators and Virtual Environments* 1(1): 127-132. March, 1992.

21. Zeltzer, D. and S. Drucker . A Virtual Environment System for Mission Planning. *Proc. 1992 IMAGE VI Conference,* Phoenix AZ.July, 1992.