Supporting Interactive Animation Using Multi-way Constraints

Jean-Francis Balaguer and Enrico Gobbetti

CRS4
Center for Advanced Studies, Research and Development in Sardinia
Scientific Visualization Group
Via Sauro 10
09123 Cagliari, Italy

E-mail: balaguer@crs4.it, gobbetti@crs4.it

To appear in

Proc. Eurographics Workshop on Programming Paradigm for Graphics (1995)
Veltkamp R and Blake E, Editors, Springer-Verlag, Vienna.

Abstract This paper presents how the animation subsystem of an interactive environment for the visual construction of 3D animations has been modeled on top of an object-oriented constraint imperative architecture. In our architecture, there is no intrinsic difference between user-interface and application objects. Multi-way dataflow constraints provide the necessary tight coupling among components that makes it possible to seamlessly compose animated and interactive behaviors. Indirect paths allow an effective use of the constraint model in the context of dynamic applications. The ability of the underlying constraint solver to deal with hierarchies of multi-way, multi-output dataflow constraints, together with the ability of the central state manager to handle indirect constraints are exploited to define most of the behaviors of the modeling and animation components in a declarative way. The ease of integration between all system's components opens the door to novel interactive solution to modeling and animation problems. By recording the effects of the user's manipulations on the models, all the expressive power of the 3D user interface is exploited when defining animations. This performance-based approach complements standard key-framing systems by providing the ability to create animations with straight-ahead actions. At the end of the recording session, animation tracks are automatically updated to integrate the new piece of animation. Animation components can be easily synchronized using constrained manipulation during playback. The system demonstrates that, although they are limited to expressing acyclic conflict-free graphs, multi-way dataflow constraints are general enough to model a large variety of behaviors while remaining efficient enough to ensure the responsiveness of large interactive 3D graphics applications.

1 Introduction

Modern 3D graphics systems allow a rapidly growing user community to create and animate increasingly sophisticated worlds. Despite their inherent three-dimensionality, these systems are still largely controlled by 2D WIMP[1] user-interfaces. The inadequacy of user-interfaces based on 2D input devices and mindsets becomes particularly evident in the realm of interactive 3D animation. In this case, the low-bandwidth communication between user-interface and application and the restrictions in interactive 3D motion specification capabilities make it very difficult to define animations with straight-ahead actions. This inability to interactively specify the animation timing is a major obstacle in all cases where the spontaneity of the animated object's behavior is important [1][9].

To explore the enormous potential of 3D interactive techniques for providing solutions to modeling and animation problems, we have developed *Virtual Studio*, an integrated 3D animation environment where all interaction is done in three dimensions and where multi-track animations are defined by recording users' manipulations on 3D models. This way, we bring the expressiveness of real-time motion capture systems into a general-purpose multi-track system running on a graphics workstation. 3D devices allow the specification of complex 3D motion, while virtual tools are visible mediators that provide interaction metaphors to control application objects. Effective editing of recorded manipulations is made possible by compacting a continuous parameter evolution with an incremental data-reduction algorithm, able to compute spline representations that preserve both geometry and timing.

In this paper, we concentrate on how we modeled the animation subsystem of *Virtual Studio*. First, we present the object model and describe the class hierarchy of the animation subsystem. Then, we show how the animation behavior is obtained using hierarchical data-flow constraints. Other aspects of *Virtual Studio* are presented elsewhere: references [13][14] presents the underlying graphics architecture (named *VB2*), references [1][2][15] provides a general overview of the animation system, and references [1][3] detail the data reduction algorithm.

2 Object Model

2.1 Primitive Elements

In *Virtual Studio*, there is no intrinsic difference between user interface and application objects. The tight integration between all the components of an animated environment (i.e. interaction, application, and animation objects) is obtained by the means of a constraint-imperative object-oriented (OOCIP) architecture [11]. In our architecture, the state of the system, the long-lived relations between state components, and the sequencing relations between states are represented by different primitive elements: *active variables*, *hierarchical constraints*, and *daemons*.

[1] WIMP stands for Window, Icon, Menu, Pointing device.

Active variables and information modules. An *active variable* maintains its value and keeps track of state changes by recording its value every time it is modified. The history of the values of each variable is limited by the user (by default, each active variable maintains only one value). All *VB2* objects are instances of classes in which dynamically changing information is defined with active variables related through hierarchical constraints. Grouping active variables and constraints in classes permits the definition of *information modules* that provide levels of abstraction that can be composed to build more sophisticated behavior.

Hierarchical constraints. The bi-directional information exchange between components required to integrate animated and interactive behaviors [6][12][14][15] is obtained with *hierarchical multi-way constraints* [5] maintaining long-lived relations between active variables. To support local propagation, constraint objects are composed of a declarative part defining the type of relation that has to be maintained and the set of constrained variables, as well as of an imperative part, the list of possible methods that could be selected by the constraint solver to maintain the constraint. A *priority level* is associated with each constraint to define the order in which constraints need to be satisfied in case of conflicts [5]. This way, both required and preferred constraints can be defined for the same active variable. Constraint methods can be general side-effect free procedures that ensure the satisfaction of the constraint, after their execution, by computing some of the constrained variables as a function of the others. Constraints are maintained using an efficient local propagation algorithm based on *Skyblue* [13][14][19], a domain-independent solver able to maintain a hierarchy of multi-way, multi-output dataflow constraints. The main drawback of such a local propagation algorithm is the limitation to acyclic onstraint graphs. However, as noted by Sannella et al. [21], cyclic constraint networks are seldom encountered in the construction of user interfaces, and limiting the constraint solver to graphs without cycles gives enough efficiency and flexibility to create highly responsive complex interactive systems. In *VB2*, introducing at runtime a constraint that would create a cyclic graph causes an exception that can be handled to remove the offending constraints[2]. The state manager behavior and the constraint solving techniques are detailed in [13][14].

Daemons. *Daemons* are the imperative portion of *VB2.* Daemons register themselves with a set of active variables and are activated each time their value changes. They are executed in order of their activation time, which corresponds to breadth-first traversal of the dependency graph. The action taken by a daemon can be a procedure of any complexity that may create new objects, perform input/output operations, change active variables' values, manipulate the constraint graph, or activate and deactivate other daemons. The execution of a daemon's action is sequential and each manipulation of the constraint graph (assignment, assertion and retraction of a constraint) advances the state manager time for recording the variable's history. State manager time and animation

[2] *VB2*'s current constraint solver [13][19] is unable to find acyclic solutions of potentially cyclic constraint graphs. An algorithm that removes this limitation is presented in [22].

time are kept separate: state manager time always goes forward and is used to keep track of the successive states of the system; animation time can instead be controlled by the user, to move backwards and forwards in a sequence, and is bound to real-time during animation recording and playback.

Variable paths. In *VB2*, daemons and constraints locate their variables through *indirect paths*. An indirect path is an object able to compute the location of a variable as well as the list of the intermediary variables that were used to compute this variable. Active variables are viewed in this context as self-referencing indirect paths using no intermediary variables. When a path is not capable of locating the variable, it is said to be *broken*. A simple example of indirect path is the *symbolic path*, which corresponds to Garnet's pointer variable [23] (an example is *parent_global_transf := Current/"parent"/"global_transf"*, where *"/"* indicates indirection, and quoted names correspond to the names of active variables in constraint modules). As stated by Vander Zanden et al. [23], the use of indirect paths allows constraints to model a wide array of dynamic application behaviors, and promotes a simpler, more effective style of programming than conventional constraints. Indirect paths are implemented in *VB2* by deactivating and reactivating constraints and daemons as soon as an intermediary variable used for computing one of their paths is modified [13], as in the user interface toolkit *Multi-Garnet* [20]. Only daemons and constraints that do not have any broken path are successfully activated. The others remain on wait until active variable modifications allow their paths to locate all the variables.

VB2 and *Virtual Studio* are implemented in the object-oriented language *Eiffel* [17]. All the primitive elements of *VB2* are modeled using abstract classes from which all the other components of the systems are derived. The class C_MODULE represents a *VB2* core object, composed of variables, constraints, and daemons. Variables are represented as instances of C_VARIABLE, daemons as instances of DAEMON, and constraints of CONSTRAINT. Selective export rules are used to solve one of the problems of the integration at the library level of constraints with an object oriented framework [4][11]. In particular, only constraint methods (instances of descendants of class C_METHOD) have the right to assign new values to active variables, and assertions control that these assignments are done exclusively during constraint propagation.

2.2 Modeling Subsystem

The central component of *Virtual Studio*'s modeling subsystem is the NODE_3D class, whose instances, related in a hierarchical fashion, represent the transformation hierarchy. Position, orientation, shearing and scaling of the reference frame are packaged in TRANSFORM_3D objects. Degrees of freedom can be attached to a node in order to define additional constrained motion, as in articulated structures. Instances of MATERIAL and TEXTURE are used to define the behavior of physical objects with respect to light. Placing instances of MATERIAL and TEXTURE in a node allows instance inheritance through the hierarchy. Instances of LIGHT represents light sources whose color and intensity are defined by instances of MATERIAL and TEXTURE. An instance of CAMERA repre-

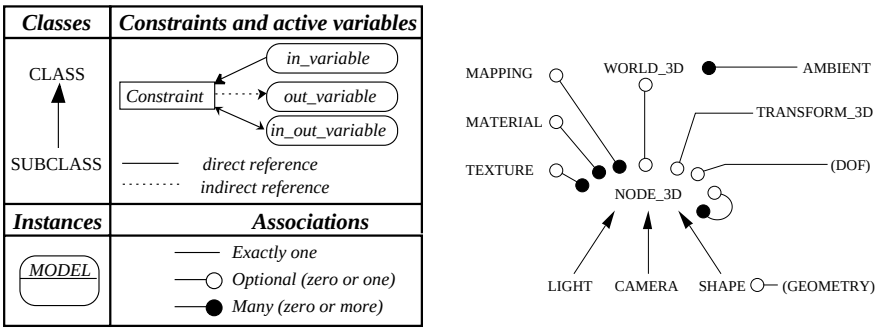

Fig. 1: Design notation and simplified modeling hierarchy.

sents a camera viewing the scene. It maintains information about its viewing frustum and a possibly stereoscopic projection. Instances of SHAPE encapsulate the concept of physical objects having a geometry, material and texture in the Cartesian space. More details on the class hierarchy of the modeling and rendering clusters are presented in [13]. A simplified object-relation diagram of the modeling subsystem as well as the design notation can be found in Figure 1.

2.3 Animation Subsystem

The animation is viewed as a time interval hierarchy where each level is represented by an instance of subclasses of the TIME_INTERVAL class. Its root is an instance of the ANIM_MANAGER class representing the overall animation. This object is responsible for maintaining the animation time (represented as active variables), which can be controlled by the user to access different parts of a sequence, and is bound to real-time during animation playback. Animation time, as opposed to the state manager time, is continuous, and frames in the animation are sampled when needed. An instance of ANIM_MANAGER is composed by a set of CONTROLLER instances. Controllers are objects allowing the animation of the parameters of a *Virtual Studio*'s graphical object. They are composed of a set of instances of TRACK, one for each animated parameters. The generic class VALUE_INTERVAL defines the concept of interval of values. Through multiple inheritance, the behaviors of TIME_INTERVAL and VALUE_INTERVAL are composed to define the HISTORY class, where each value of the value interval is put in correspondence with a time in the time interval. The subclass TRACK represents a track of values as a function of animation time, and is defined as a list of instances of subclasses of the BLOCK class. The subclass CONTINUOUS_BLOCK defines a history of continuous values, represented using a parametric B-spline curve [10] (instances of P_SPLINE). The subclass DISCRETE_BLOCK defines a history of discrete values represented as the successive value changes in time. Discrete history values can be of any type, from atomic types to aggregate objects. Finally, the subclass CONSTANT_BLOCK defines a history with a constant value over the time interval. The controller uses

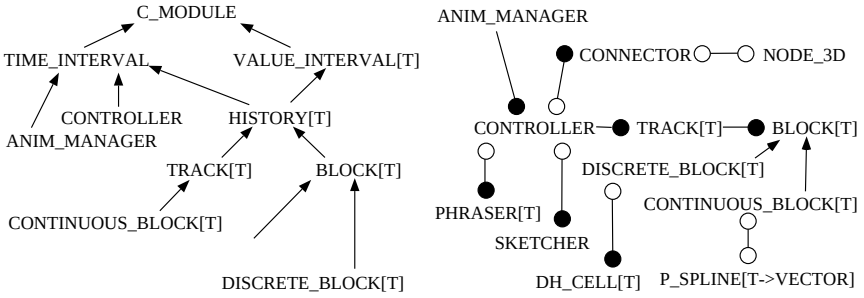

Fig. 2: Class hierarchy and object-relation diagram of the animation subsystem.

SKETCHER and PHRASER objects to handle continuous tracks. SKETCHER objects apply an incremental data reduction algorithm [1][3] to convert a series of sampled values of a variable to a B-spline representation that is then stored in instances of CONTINUOUS_BLOCK. PHRASER objects are used to blend animations defined in adjacent blocks using Hermite spline segments that join the blocks with C^1 continuity [1][10]. The class hierarchy and the relation between instances of the animation subsystem are presented in Figure 2.

2.4 Controller-Model Protocol

Animation recording and playback is obtained by binding controllers to models. When *binding* a model to a controller, the controller must first determine if it can animate the given model, identifying on the model the set of public active variables requested to activate its binding constraints. Once the binding constraints are activated, the model is ready to be animated. The binding constraints being generally bi-directional, the controller is always informed of model's information changes even if it is modified by other objects, and conversely, during animation playback, the state of the model is modified to reflect the tracks' state changes. *Unbinding* a model from a controller detaches it from the object it animates. The effect is to deactivate the binding constraints in order to suppress dependencies between controller's and model's active variables. Once the model is unbound, it does not participate to the animation and its state remains unchanged during playback. When the user manipulates the model, the constraint network is oriented from the model to the tracks, while during animation playback it is oriented in the opposite direction (from the tracks to the model). This behavior is obtained by associating to interaction constraints a priority higher than that of playback constraints.

Indirect constraints allow to define a controller's binding mechanism entirely in a declarative way. In Figure 3, the connector object defines the binding with equality constraints between the camera's and the controller's state variables. These variables are located with indirect paths that use the connector's ports as intermediary variables. Second-order control is used inside the connector to ensure that all binding constraints are bound or unbound simultaneously [13].

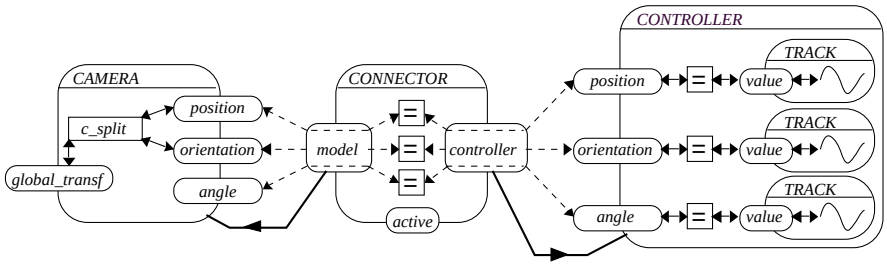

Fig. 3: Animating a model is obtained by binding a controller.

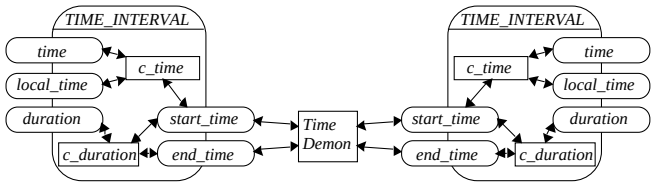

Fig. 4: Simplified constraint network for instances of TIME_INTERVAL.

3 Animation

3.1 Maintaining the Temporal Coherence

All objects involved in the definition of the temporal subdivision of the animation are instances of subclasses of the abstract class TIME_INTERVAL, which defines the behaviors used to maintain a time interval. The *start_time* and *end_time* active variables of TIME_INTERVAL instances store the lower and upper limits of the interval, expressed in absolute animation time. The *time* and *local_time* active variables stores the current time value expressed, respectively, in absolute animation time and track local time.

Daemons depending on the *start_time* and *end_time* variables of each time interval object, are responsible for the propagation of timing modifications between the levels of the hierarchy, so as to permanently maintain the temporal coherence between the hierarchy levels. That way, the animation timing can be manipulated globally or locally. For example, changing the duration of a single track will propagate up in the hierarchy so as to modify the total duration of the animation. Conversely, changing the duration of the overall animation will propagate down in the hierarchy and apply a scale factor to subdividing intervals. Modifying an intermediate level will propagate down to scale its subdividing intervals and up to update the total duration of the animation. Figure 4 shows the constraint network formed by two levels of the time interval hierarchy.

Since the temporal coherence of the time interval hierarchy is being permanently maintained, it is possible to declaratively specify synchronizations between tracks by introducing constraints between their timing variables. When tracks are made visible, the start and end time variables can be manipulated

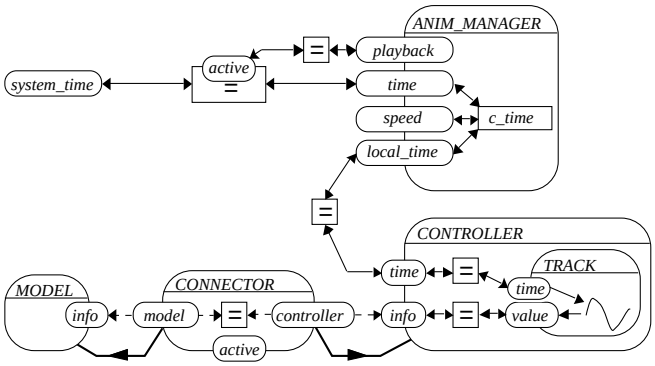

Fig. 5: Time propagation during animation playback triggers model updates.

through the associated binders. Figure 7 (see color plates) shows an animated camera tool together with the camera position track. Synchronizations between the evolution of different parameters may be obtained by interactively connecting together time binders of the associated tracks. In the figure, the start binder is represented by the 3D widget on the left, which has a value of 0. 6, while the end binder is on the right and has a value of 4.0. Start and end times can be interactively controlled by selecting one of these widgets with the 3D cursor and dragging a line to a time binder of the object with which we want to synchronize the animation. Internally, this operation enforces an equality constraint between the active variables controlled by the binders.

3.2 Playing the Animation

One of the basic features of an interactive animation system is to provide real-time interactive animation playback. This is obtained by evaluating all animation tracks at the desired time, updating the scene's state and rendering the frame. The following frame time is computed by taking into account the time needed to generate the previous frame in the animation. Animation can thus be played in real-time and synchronized with external data sources. This is possible because the animation time is continuous and frames appear only as a result of sampling operations. Animation playback is obtained by having the values of the animation time control the sampling time of each of the tracks in the system. The beginning and end of the animation playback are indicated by modifying the value of a Boolean variable in the animation manager. This triggers the activation or deactivation of a constraint between the system's time and the animation manager's time. Once the constraint is activated, time changes propagate through the time interval hierarchy triggering the tracks' evaluation and the models' update (see Figure 5).

Evaluating a continuous parameter track involves determining which block is active at the current time and asking its value. If the current time falls into the transition period between two successive blocks, then the value is computed

by a phraser object whose behavior is to blend the value of successive blocks in order to ensure a smooth transition. Discrete tracks are defined by the successive transitory values together with the times at which the value change events occur. The value of the track is the value of the event coming before the time at which the track is evaluated. To allow track evaluation at any time, evaluation requests at times before the track's start time or after the track's end time return the value at, respectively, start and end time.

3.3 Recording the Animation

In *Virtual Studio*, animation is specified with a performance-based approach by recording the effects over time of the user's manipulation on the models. A 3D cursor, controlled by a *Spaceball*, is used to select and manipulate objects of the synthetic world. Direct manipulation and virtual tools are the two techniques used to input information. Both techniques involve using mediator objects that transform cursor's movements into modifications of manipulated objects. Virtual tools are visible first class objects that lie in the same 3D space as application objects and offer the interaction metaphor to control them. Their visual appearance is determined by a modeling hierarchy, while their behavior is controlled by an internal constraint network [14]. As in the real world, the user configures its workspace by selecting tools, positioning and orienting them in space, and binding them to application objects. At the moment of binding, the tool's and the application object's constraint networks become connected, so as to ensure information propagation. When bound, the tool changes its visual appearance to a shape that provides information about its behavior and offers semantic feedback. Multiple tools can be active simultaneously in the same 3D environment in order to control all its aspects. The environment's consistency is continuously ensured by the underlying constraint solver. The bi-directionality of the relationships between user-interface and application objects makes it possible to use virtual tools to interact with a dynamic environment, opening the door to the integration of animation and interaction techniques.

During manipulation, the internal constraint networks of the user-interface mediator object are connected to the 3D cursor and to the manipulated model with binding constraints. If the manipulated object is animatable, binding constraints are also in place to connect the manipulated model to its controller. The user becomes thus the source of a flow of information that propagates through the internal constraint networks of the user-interface mediator object, of the manipulated object and of the animation controller (see Figure 6). In order to be able to record the evolution over time of the model's information during manipulation, each track owns a recording daemon whose task is to store the variations of the monitored variable and to update the track at the end of the manipulation. For continuous tracks, the data reduction algorithm is applied to the incoming data and the approximation spline is inserted in the track. Discrete tracks are built from the successive transitory value changes triggered by the user's manipulations. Multiple recording daemons may be active simultaneously so as to record the variations of all variables influenced by the manipulation. That way,

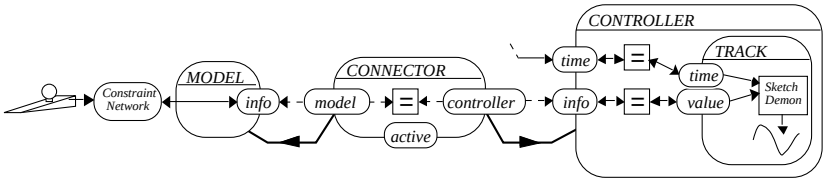

Fig. 6: While recording, device sensor values are connected to tracks' variables.

it is possible to interact with the animated object at the task level [16], as the 3D motion described by the 3D cursor can be interpreted as a high-level goal allowing the simultaneous and coordinated control of several parameters as, for example, when guiding a walking articulated figure.

Since interaction constraints have a higher priority than the binding constraints of the controllers, recording can occur during animation playback. For a given model, the tracks that are not influenced by the interaction participate in the animation playback, while the variations of the variables being modified by the interaction are recorded. The recording daemons are able to determine when to record the modifications of the variable they monitor by analyzing the local orientation of the constraint network that binds the controller to the model: an incoming constraint network means that the variable is modified due to the user's manipulation, and not by animation playback. This behavior is obtained in a declarative way by exploiting the ability of the constraint solver to deal with hierarchies of dataflow constraints. It allows the system to promote the use of a layered approach to animation specification, where the user starts by recording some part of the animation, and later defines additional pieces that are automatically synchronized with the rest of the animation by manipulating the models during animation playback.

3.4 Example

The example scene (see Figure 9 in color plates) is composed of a character, a light and a camera. An appropriate virtual tool and a controller object has been connected to each scene element to provide support for manipulation and animation. A tool encapsulating a walking engine has been bound to the character to provide the model with a walking behavior. During manipulation or animation, the tool is responsible to generate the walking cycle animation according to the translation speed. Lookat constraints make the camera and the light always point towards the character head. During animation playback, the camera and the light positions are determined by their recorded paths, while their orientations are determined by the lookat constraints. By connecting together the start and end time binders of the character's and light's tracks, we introduced synchronization constraints so that both motions perform in parallel. The camera motion has been made to start after the character's motion by interactively connecting the camera's start time binder with the character's end time binder.

The environment presented in Figure 8 (see color plates) is composed of

5632 constraints and 13659 active variables. The scene's graphical representation contains 3000 polygons illuminated by a spot light (the animated light source) and a directional light (a light attached to the user's viewpoint). The redraw rate was 10 frames per second on a *Silicon Graphics Crimson VGX*. Despite the complexity of the constraint network, rendering speed was largely the limiting factor, since 80% of the application time was spent in rendering operations.

4 Conclusion

In this paper, we have presented how the animation subsystem of *Virtual Studio* has been modeled on top of the OOCIP *VB2* architecture. In *VB2*, there is no intrinsic difference between user-interface and application objects. Multi-way dataflow constraints provide the necessary tight coupling among components that makes it possible to seamlessly compose animated and interactive behaviors. Indirect paths allow an effective use of the constraint model in the context of dynamic applications. The ability of the underlying constraint solver to deal with hierarchies of multi-way, multi-output dataflow constraints, together with the ability of the state manager to handle indirect constraints, are exploited to define most of the behaviors of the modeling and animation components in a declarative way. The ease of integration between all system's components opens the door to novel interactive solution to modeling and animation problems. By recording the effects of the user's manipulations on the models, all the expressive power of the 3D user interface is exploited when defining animations. This performance-based approach complements standard key-framing systems by providing the ability to create animations with straight-ahead actions. At the end of the recording session, animation tracks are automatically updated to integrate the new piece of animation. Animation components can be easily synchronized using constrained manipulation during playback.

Our system demonstrates that, although they are limited to expressing acyclic conflict-free graphs, multi-way dataflow constraints are general enough to model a large variety of behaviors, while remaining efficient enough to ensure the responsiveness of large interactive 3D graphics applications. In the graphics community, these techniques have until recently been largely confined to 2D applications [7][18]. To our knowledge, *VB2* and *TBAG* [8] are the first 3D graphics systems that uniformly use multi-way constraint networks to model large animated environments.

Acknowledgments

The authors would like to thank Ronan Boulic for providing the walking engine software, and the workshop reviewers for their helpful comments and suggestions.

This research was conducted while the authors were at the Swiss Federal Institute of Technology in Lausanne.

References

[1] Balaguer JF (1993) *Virtual Studio*. PhD Thesis, EPFL, Switzerland.

[2] Balaguer JF, Gobbetti E (1995) Animating Spaceland. *IEEE Computer* 28(7).

[3] Balaguer JF, Gobbetti E (1995) Sketching 3D Animations. *Proc. EUROGRAPHICS*: 241-258.

[4] Blake E, Hoole Q (1992) Expressing Relationships between Objects: Problems and Solutions. *Proc. Third EUROGRAPHICS Workshop on Object-Oriented Graphics*: 159-162.

[5] Borning A, Freeman-Benson B, Wilson M (1992) Constraint Hierarchies. *Lisp and Symbolic Computation* 5(3): 221-268.

[6] Conner DB, Snibbe SS, Herndon KP, Robbins DC, Zeleznik RC, Van Dam A (1992) Three-Dimensional Widgets. *Proc. SIGGRAPH Symposium on Interactive 3D Graphics*: 183-188.

[7] Duisberg R (1986) *Temporal Constraints in the Animus System*. PhD Thesis, TR-86-09-01, Computer Science Department, University of Washington.

[8] Elliott C, Schechter G, Yeung R, Abi-Ezzi S (1994) TBAG: A High Level Framework for Interactive, Animated 3D Graphics Applications. *Proc. SIGGRAPH*: 421-434.

[9] Elson M (1993) *Character Motion Systems*. SIGGRAPH Course Notes 1.

[10] Farin G (1990) *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press.

[11] Freeman Benson B (1990) Mixing Objects, Constraints, and Imperative Programming. *Proc. ECOOP/OOPSLA*: 77-87.

[12] Gleicher M (1993) A Graphics Toolkit Based on Differential Constraints. *Proc. UIST*: 109-120.

[13] Gobbetti E (1993) *Virtuality Builder II*. PhD Thesis, EPFL, Switzerland.

[14] Gobbetti E, Balaguer JF (1993) VB2: A Framework for Interaction in Synthetic Worlds. *Proc. UIST*: 167-178.

[15] Gobbetti E, Balaguer JF (1995) An Integrated Environment to Visually Construct 3D Animations. *Proc. SIGGRAPH*: 395-398.

[16] McKenna M, Pieper S, Zeltzer D (1990) Control of a Virtual Actor: The Roach. *Proc. SIGGRAPH Symp. on Interactive 3D Graphics*: 165-174.

[17] Meyer B (1993) *Eiffel: The Language*. Prentice-Hall.

[18] Myers BA, Giuse GA, Dannenberg RB, Vander Zanden B, Kosbie DS, Pervin E, Mickish A, Marchal P (1990) Garnet: Comprehensive Support for Graphical, Highly-Interacive User Interface. *IEEE Computer* 23(11): 71-85.

[19] Sannella M (1994) Skyblue A Multi-Way Local Propagation Constraint Solver for User Interface Construction. *Proc. UIST*: 137-146.

[20] Sannella M, Borning A (1992) *Multi-Garnet: Integrating Multi-way Constraints with Garnet*. TR-92-07-01, Dept. of Computer Science, University of Washington.

[21] Sannella M, Maloney J, Freeman Benson B, Borning A (1992) Multi-way versus One-way Constraints in user-Interface Construction. *Software Practice and Experience* 23(5): 529-566.

[22] Vander Zanden B (1995) *An Incremental Algorithm for Satisfying Hierarchies of Multi-Way, dataflow Constraints*. Technical Report, University of Tennesse, Knoxville.

[23] Vander Zanden B, Myers BA, Giuse D, Szeleky P (1991) The Importance of Pointer Variables in Constraint Models. *Proc. UIST*: 155-164.

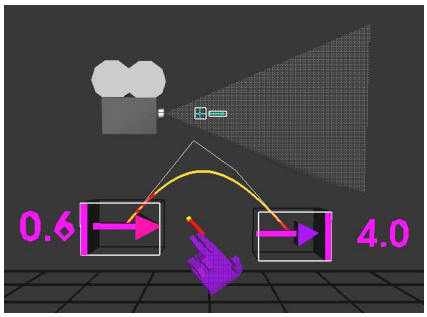

Fig. 7: Camera tool and camera position track.

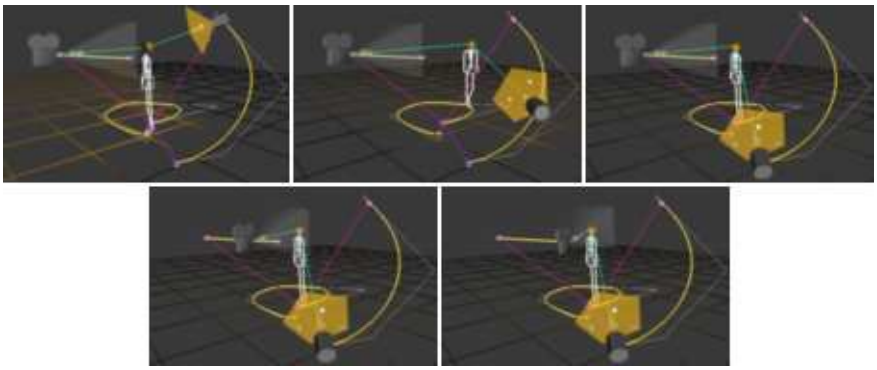

Fig. 8: The scene is composed of a character, a light and a camera.