Animating Spaceland[1]

Jean-Francis Balaguer
Enrico Gobbetti

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

E-mail: {balaguer | gobbetti}@crs4.it

DRAFT SUBMISSION

IEEE Computer
Appeared as "3D User Interfaces for General-Purpose 3D Animation"
29(8) August 1996

Abstract

Modern 3D animation systems allow a rapidly growing community to create and animate increasingly sophisticated worlds. Despite the inherent three-dimensionality of these tasks, the user interfaces of such systems are still predominantly two-dimensional, using 2D input devices and techniques which severely limit the range of tasks that can be accomplished interactively. In particular, these limitations make it very difficult to interactively input complex 3D movements. In this paper, we present *Virtual Studio*, a 3D animation environment where all the interaction is done directly in three dimensions. 3D devices allow the specification of complex 3D motion while virtual tools are visible mediators that provide interaction metaphors to control application objects. An underlying constraint solver, that automatically maintains multi-way relationships, provides the ability to tightly couple application and interface objects. The animation is defined by recording the effect of the user's manipulations on the models, taking into account the temporal aspect of the interaction. Data reduction techniques are applied to obtain editable representation of continuous parameters' evolution.

Keywords

3D Interaction, 3D Virtual Tools, Virtual Environment, 3D Animation, Object-Oriented Graphics, Hierarchical Constraints, Data Reduction.

[1]See Edwin Abbott, *FLATLAND, a Romance of Many Dimensions*, Basil Blackwell, Oxford.

1. Introduction

The continued improvement and proliferation of graphics hardware for workstations and personal computers has brought increasing prominence to a newer, dynamic style of software application program. Highly interactive graphical interfaces that allow the presentation and the direct manipulation of information in a pictorial form are now an important part of most of modern software applications, and are often considered essential for making computers easy to use. The latest graphics workstations have become true digital media platforms that combine sound generation, interactive 3D graphics and movie playback capabilities. The relatively low cost of these workstations, comparable to that of high-end personal computers, makes it possible for a large community of users to benefit from this evolution. Multi-media authoring tools allow the creation of increasingly sophisticated presentations that effectively convey ideas. Hyper-media documents are increasingly being used to build multi-media tutorials, catalogs or guided tours where users can interactively access information under various forms, from simple text and static images to animated sequences with sounds. With interactive *CD* and interactive digital cable television, hyper-media documents will be brought to a wide audience directly through their TV set. Computer animation and multi-media are also being used more and more as a means to facilitate learning in such diverse areas as architecture, biology, mathematics, art, and computer science. The rapid development of these new diffusion channels requires the production of a large numbers of animated sequences, a growing portion of which will be devoted to 3D computer graphics movies [12]. These animations need to be produced rapidly, at low cost, and often by people working alone who are not professional animators.

1.1 Current animation systems

A number of professional 3D animation systems are currently available for use on desktop graphics workstations [11]. The goal of most of these systems is to offer comprehensive modeling, rendering, and animation capabilities for the production of commercial special effects or animation. Even if such systems can be used to generate low-cost short animation sequences, such as those in tutorials or multi-media documents, their use for this task has several drawbacks.

First of all, the large amount of functionality that these systems must offer, in order to be capable of producing high-quality movies, makes them too complex and too expensive to be used by non-professional animators for the creation of short sequences. Taking advantage of the potential of these animation systems clearly requires time and effort to understand them and to develop technical skills [11]. If such efforts are well spent for professional animators, they represent too large an investment for infrequent users.

The inadequacy of current systems' user interfaces is another reason why they are not well suited to the rapid production of animation sequences of limited complexity. Despite the inherent three-dimensionality of these tasks, such systems still predominantly use 2D input devices and techniques [5]. Traditional 2D widgets are the most common way of interacting with the objects that comprise the simulated world, but the complete separation between interface and application offered by this solution makes it difficult for the user to correlate the manipulation with its resulting effect on the object. In such systems, direct interaction with the synthetic world is generally limited to interactive viewing, selecting or positioning of objects, and the specification of 3D information is normally obtained by gestural interpretation of mouse motion. The limitation of the device to two dimensions reduces the range of possible movements that can be specified in a single interaction task and forces users to change program and mental modes in order to input spatial information.

This kind of user interface does not take advantage of 3D space and puts severe limitations on the tasks that can be accomplished interactively in a graphics application. In particular, the users of an interactive 3D animation system are generally confined to the specification of key postures, which are then interpolated to produce smooth motion, since the limitations of the interface make it very difficult to interactively input complex 3D movements. This inability to specify the timing of an animation in an interactive way is a major drawback in all cases where the spontaneity of the animated object's behavior is important [6][9].

1.2 Towards a new generation of animation systems

In order to provide animators with tools allowing them to create animations by straight-ahead actions, and not only by using pose-to-pose techniques, we have to change the way the interface of current graphics systems is conceived. Recent research work has shown how high-bandwidth interaction with 3D objects and environments can be obtained for virtual environments [8] and performance animation systems [9] by means of devices allowing the control of multiple degrees of freedom. More conventional configurations can be used as well, together with applications whose interaction metaphors exploit the greater possibilities of 3D [4][5][10]. However, very few attempts have been made to apply the results of this research to enhance the animation capabilities of current general-purpose animation systems. Performance animation systems, built around custom multiple degrees of freedom devices for motion capture, are not general enough to define all aspects of an animation, while the advanced 3D interaction capabilities of other systems are currently confined to modeling aspects.

In this paper, we describe *Virtual Studio*, an integrated 3D animation environment designed to provide all the interactivity of a performance animation approach to a general-purpose multi-track system. *Virtual Studio* exploits the expressiveness of state-of-the-art 3D interfaces to allow interaction with all aspects of synthetic worlds entirely in three dimensions (see figure 1). 3D devices allow the specification of complex 3D motion, while virtual tools are visible mediators that provide interaction metaphors to control application objects. An underlying constraint solver, that

automatically maintains multi-way relationships, provides the ability to tightly couple application and interface objects. The expressive power of this user interface paradigm makes it possible to define a large variety of animations by recording the effects and the timing of user manipulations on models. Data reduction techniques are automatically applied to obtain editable representations of continuous parameter evolution, overcoming in this way one of the major limitations of current performance animation systems [9].

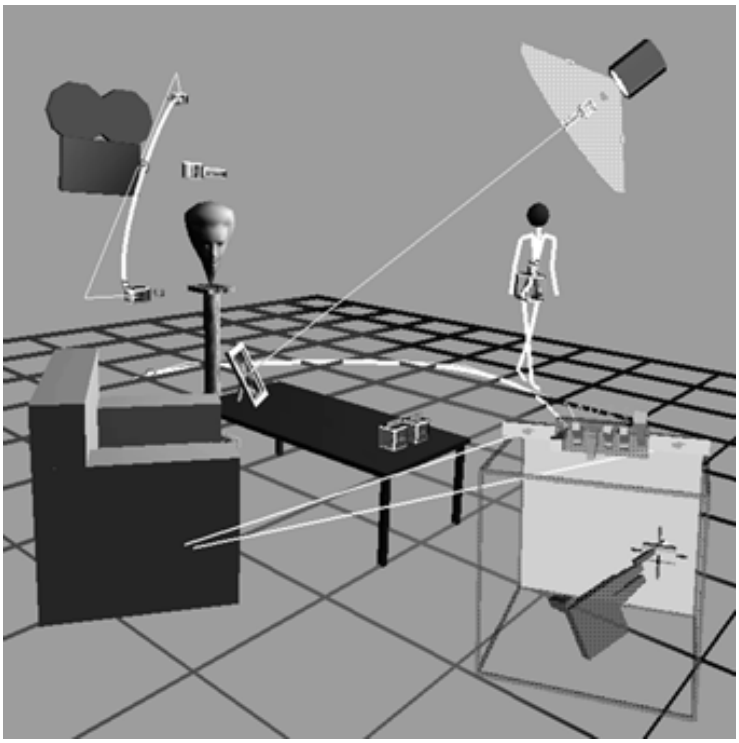


Figure 1. *Virtual Studio*

The remainder of the paper describes the various aspects of *Virtual Studio*, with an emphasis on the interactive definition of complex 3D animations. First, we will give an overview of the system. Next, we will concentrate on how the user interacts with a synthetic environment through direct manipulation and virtual tools, and on how these techniques can be used to interactively define animations. The system's capabilities are then illustrated by an example. The paper concludes with a discussion of the results obtained and a view of future work.

2. System Overview

Virtual Studio is an animation environment built on top of *VB2* [3][4], a graphics architecture based on objects and constraints. Its goal is to allow the manipulation and the animation of all aspects of synthetic worlds entirely in three dimensions.

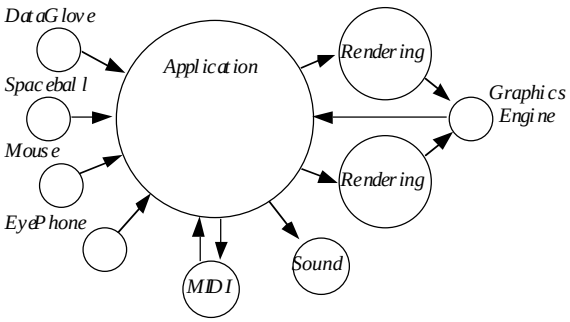


Figure 2. Overall structure of *VB2* applications

A *VB2* application is composed of a group of processes communicating through inter-process communication (IPC). Figure 2 shows the typical configuration of an interactive application. Processes are represented as circles, while arrows indicate the information flow between them. Each of the processes is continuously running, producing and consuming asynchronous messages to perform its task. A central application process manages the model of the virtual world, and simulates its evolution in response to events coming from other processes which read the input device sensors at specified frequencies. Sensory feedback to the user can be provided by several output devices. Visual feedback is

provided by real-time rendering on graphics workstations, while audio feedback is provided by MIDI output and playback of prerecorded sounds. The application process is by far the most complex component of the system. This process has to respond to asynchronous events by making the virtual world's model evolve from one coherent state to the next and by triggering appropriate visual and audio feedback.

The following sections briefly describe the device configurations used and the dynamic model of the application process. References [3] and [4] provide more details on these subjects.

2.1 Device Configurations

In order to interact with three-dimensional environments in an effective way, the user should be able to directly specify 3D motions and should be provided with enough depth cues to understand the structure of the 3D world [8]. For this purpose, we use two hardware configurations built out of off-the-shelf equipment, which are described in the following sections.

2.1.1 Immersive Configuration

The goal of *Virtual Studio*'s immersive configuration is to convince the users that they are part of the world they manipulate (figure 3). Visual feedback is provided by rendering a pair of stereo images on a single graphics workstation connected to a *VPL Eyephone* through a custom-made image splitter. The 3D input device used for manipulation is the *VPL DataGlove*. Both the *Eyephone* and the *DataGlove* use a *Polhemus Isotrak* device to track user's motion. Posture recognition based on finger flexion is used to let the user input commands and categorical information [3].

Figure 3. Immersive configuration

2.1.2 Desktop Configuration

Virtual Studio's desktop configuration uses a *Spaceball* and a mouse as input devices, and *LCD* shutter glasses for binocular perception of the synthetic world. The *Spaceball* is used for the continuous specification of spatial transformations, while the mouse is used as a picking device. Both user's hands can therefore be used simultaneously to input information (figure 4). Since one of our research goals is to explore all the possibilities of 3D interaction, we do not provide any two-dimensional interface. Keyboard commands are used instead of hand postures to change the visibility of objets and to trigger animation playback.

Although this configuration does not fully provide the illusion of immersion, we usually find it more effective for our application than the immersive one, mainly because of the low quality of our virtual reality equipment [1][3]. In particular, the low resolution of the *Eyephone* and the small working volume of the *Polhemus* trackers do not permit a comfortable interaction with complex 3D scenes.

Figure 4. Desktop configuration

2.2 Dynamic model

During interaction, the user is the source of a flow of information propagating from input device sensors to manipulated models. In order to obtain animated and interactive behavior, the system has to update its state in response to changes initiated by sensors attached to asynchronous input devices such as timers or trackers. The application is represented as a network of interrelated objects whose behavior is specified by the actions they take in response to changes in other objects on which they depend. The maintenance of the relationships between objects is delegated to a constraint-based change propagation mechanism. As presented in [3], the various aspects of the system's state and behavior are represented using different primitive elements:

- *active variables* are used to store the state of the system;
- domain-independent *hierarchical constraints* [2], are used to declaratively represent long-lived multi-way relations between active variables;
- *daemons* are used to react to variable changes for imperatively sequencing between different system states.

In this way, imperative and declarative programming techniques can be freely mixed to model each aspect of the system with the most appropriate means. As explained in [3], daemons and constraints locate their variables through *indirect expressions*, so as to allow an effective use of a constraint model in the context of dynamic applications. A central state manager is responsible for adding, removing, and maintaining all active constraints using an efficient local propagation algorithm, as well as managing the system time and activating daemons. A priority level is associated with each constraint to define the order in which constraints need to be satisfied in case of conflicts. This way, both required and preferred constraints can be defined for the same active variable. Reference [3] provides a detailed presentation of the state manager behavior and of the constraint solving techniques.

3. Interaction

In *Virtual Studio*, the users interact with simulated scenes entirely in three dimensions. A 3D cursor, controlled by the *Spaceball* or the *DataGlove* using a *node-in-hand* metaphor, is used to select and manipulate objects in the synthetic world. Direct manipulation and virtual tools are the two major techniques used to input information. Both techniques involve using mediator objects that transform the motion of the cursor into modifications of the manipulated objects. As we will see later, the true three-dimensionality of this interface is a key feature when it comes to defining animations.

3.1 Direct manipulation

The direct manipulation of aspects of three-dimensional objects is obtained by attaching constraints which directly relate the 3D cursor's active variables to variables in the dynamic model. While the interaction constraints remain active, the user can manipulate the model through the provided metaphor. The deactivation of the interaction constraints terminates the direct manipulation. Second-order constraints that depend on Boolean state variables are generally used to trigger activation and deactivation of interaction constraints. This interaction technique is used, for example, when grabbing 3D objects. While grabbed, the objects are constrained to follow the motion of the cursor. Since the cursor is controlled by a 6 degrees of freedom device, the user can interactively define complex 3D paths.

3.2 Virtual tools

The system can guide the user to understand a model's behavior and interaction metaphors by using mediator objects that present a selective view of the model's information and offer an interaction metaphor to control this information. We call these objects *virtual tools* [4].

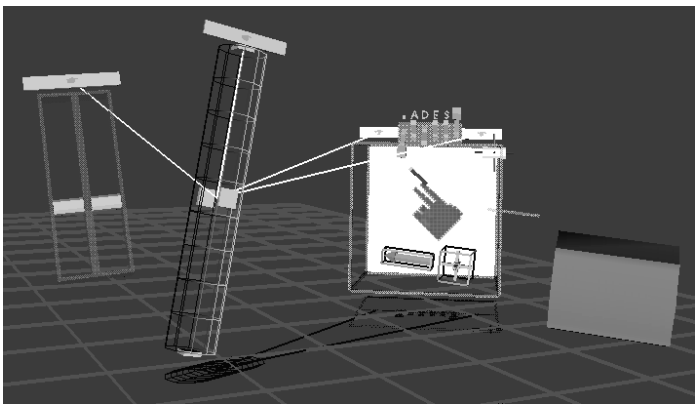


Figure 5. Virtual tools

Virtual tools are first class objects, like the widgets of *UGA* [5], which encapsulate a visual appearance and a behavior

to control and display information about application objects. The visual appearance of a tool must provide information about its behavior and offer visual semantic feedback to the user during manipulation. The tool's behavior must ensure the consistency between its visual appearance and the information about the model being manipulated, as well as allow information editing through a physical metaphor. The tool's behavior is defined by an internal constraint network, while the information required to perform the manipulation is represented by a set of active variables. The models that can be manipulated by a tool are those whose external interface matches that of the tool. The visual appearance is described using a modeling hierarchy.

In *Virtual Studio*, virtual tools are fully part of the synthetic environment (figure 5). As in the real world, the user configures his workspace by selecting tools, positioning and orienting them in space, and binding them to the models he intends to manipulate. The binding action consists in selecting an active part of the tool, called *binder* and identified by a "pointing hand" icon, and then dragging a line from the tool to the model. At the moment of binding, the tool changes its visual appearance from the box that represents it in its unbound state to a shape that provides information about its behavior and offers semantic feedback during manipulation. When the user binds a tool to a model, he initiates a bi-directional information communication between these two objects through the activation of the tool's binding constraints which maintain the coherence between the model's and the tool's information (see figure 7). Unbinding a tool from a model, expressed by dragging the line from the tool's binder to the empty space, detaches it from the objects it controls by deactivating the binding constraints, changes its visual appearance back to the unbound version, and moves it back to a default place, called the *Shelf*. Figure 6 illustrates the different interaction steps.

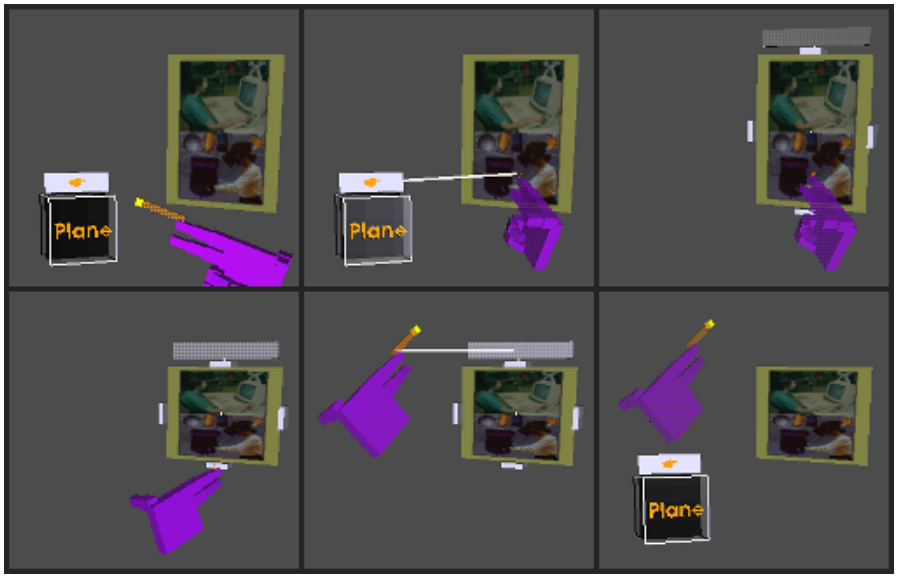
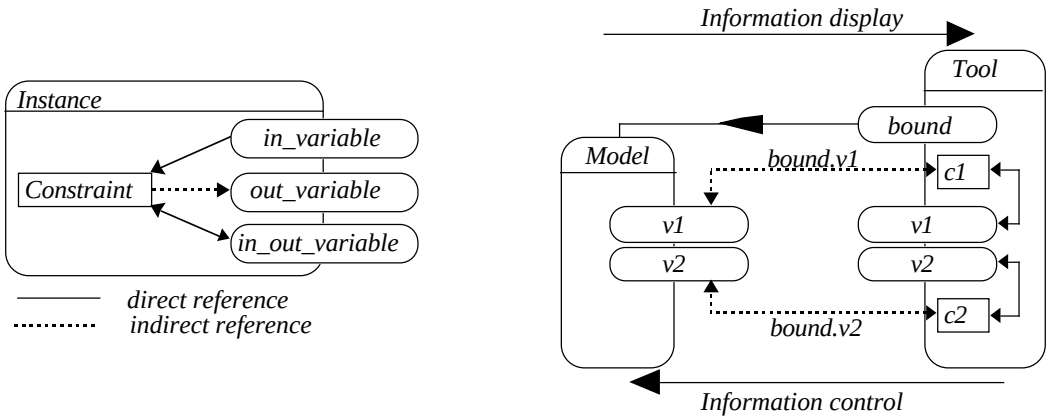


Figure 6. Interactively binding, manipulating and unbinding a tool.



(a) (b)

Figure 7a. Design notation
Figure 7b. Model and virtual tool

Virtual tools are not limited to simple editing tasks, but provide a mean to encapsulate any kind of behavior. Multiple binders may be used to connect tools to several models in order to realize complex behaviors that require information from more than a single object, as when constraining transformation components (see figure 8). Thanks to bi-directional constraints, multiple tools may be attached to a single model in order to simultaneously manipulate different parts of the model's information, or the same parts using multiple interaction metaphors. The underlying constraint solver

automatically ensures the correct behavior of the application, therefore freeing the programmer from the tedious task of maintaining dependency relationships by hand.

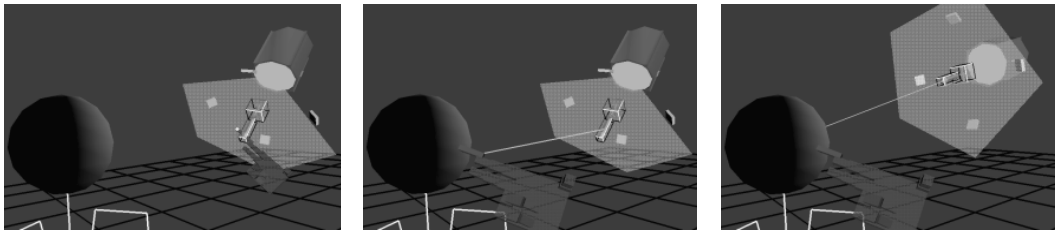


Figure 8. At the top of the image we see a *light tool*, a composite tool that displays and controls the information contained in a light source. The tool is composed of a cone tool, which controls the light projection, a body, that allows to control its position and orientation by grabbing and moving it, and of two binders (the objects in front of the light) that make it possible to connect the tool to other objects so as to constrain the light's position or orientation. For example, by connecting the arrow-shaped binder of the light tool to a model, we force the light source to always point towards the selected object.

4. Animation

Virtual tools and 3D devices provide the expressive power required for continuous control of many aspects of the synthetic world and for specification of three-dimensional motion. The system can exploit the virtual tool's full range of interaction techniques and behaviors to define animations of discrete and continuous attributes. This is done by recording the effects of the user's manipulations and taking into account the temporal aspect of the interaction. In this way, the animator has continuous control over the animation shape and timing, while key parameter based techniques offer control only at a limited number of points. Key parameter techniques, also available in *Virtual Studio*, will not be discussed here.

Recording the evolution of discrete parameters simply involves the insertion of a change value event in a temporal track. When defining the animation of continuous parameters, the amount of data collected by recording the interactive session is much too large to allow effective editing. 3D devices, being sampled typically between 10 and 30 Hz, recording a ten seconds animation, will result in the accumulation of one to three hundreds samples for each parameter influenced by the manipulation. This problem has to be faced each time some continuous information is captured by sampling the user's motion with a device, and in particular in performance animation systems [1][8]. In order to allow effective editing, a more compact representation of continuous parameter evolution must be built out of the interactive input data.

Data reduction or curve fitting techniques have been successfully applied in drafting and modeling tools for the interactive specification of 2D or 3D curves or surfaces [1]. In an animation system, since both the geometry and timing of the interactive plots need to be preserved, data reduction must be performed simultaneously in space and time. Moreover, general purpose animations need to control the evolution of parameters of various dimensions with different levels of continuity. In *Virtual Studio*, we use an incremental version of the Lyche and Mörken algorithm [7] to re-represent the interactive plot of parameter evolution with an n-dimensional B-spline, parameterized on time, which preserves the geometry and timing of the initial data. Since the spline's control polygon is defined by a much smaller number of points, the animation can be effectively edited with standard spline editing tools. Our incremental algorithm reduces the initial data by considering successive portions that are spliced together preserving continuity up to the second derivative. By putting constraints on the start and end tangents of the portion being reduced, we are able to compute the approximation using only local information [1]. In this way, the maximum time and the memory requirements of reducing a portion are constant for a given number of samples. Data reduction may therefore be performed concurrently with interactive parameter input, and the responsiveness of the application can be ensured when handling animations defined by any number of samples.

Continuous control and data reduction let users hand sketch the animation of continuous parameters much in the same way curves can be hand drawn in most drafting tools. The mediation of virtual tools makes it possible to sketch the evolution of geometric as well as non geometric attributes, while constrained or free motion can be easily specified with 3D devices. Since these devices offer continuous and simultaneous specification of three-dimensional transformations, subtle synchronizations between the position and orientation components of an animation path can be directly specified.

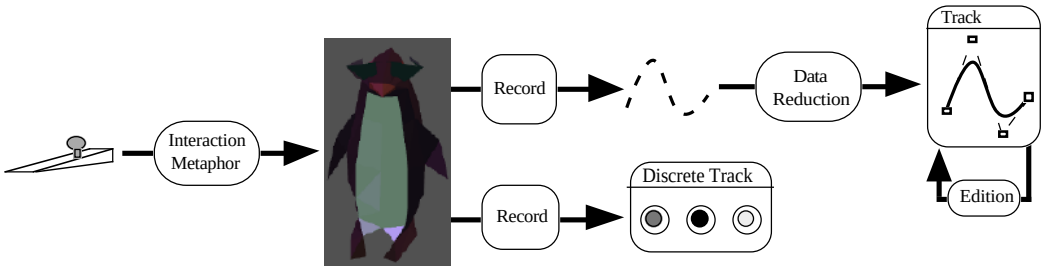


Figure 9. Interactive animation specification

In *Virtual Studio*, animations are defined by first expressing the desire to record interactive parameter evolution. A controller object is connected to each animatable model and is responsible for monitoring model state changes. While recording, all changes are handled by the controller to feed the animation tracks. Continuous tracks apply the data reduction algorithm to the incoming information, while discrete tracks simply store a change value event (see figure 9). Each track is represented as a time interval, whose start and end time variables may be constrained to specify animation synchronization. When tracks are made visible, the start and end time variables can be manipulated through the associated binders. Figure 10 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.

During playback, information propagates from the animation tracks through the controllers and down to the models (see figure 11). All connections are realized by bi-directional constraints. Since the priority of playback constraints is lower than that of interaction constraints, the user can take interactive control over animated models during playback. Therefore, animations which take into account the evolution of the environment and which present complex synchronizations with the animation of other models can be easily specified by interacting with the system during playback.

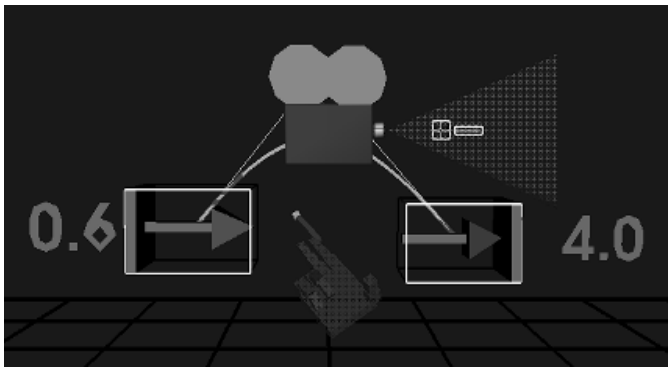


Figure 10. Camera tool and camera position track

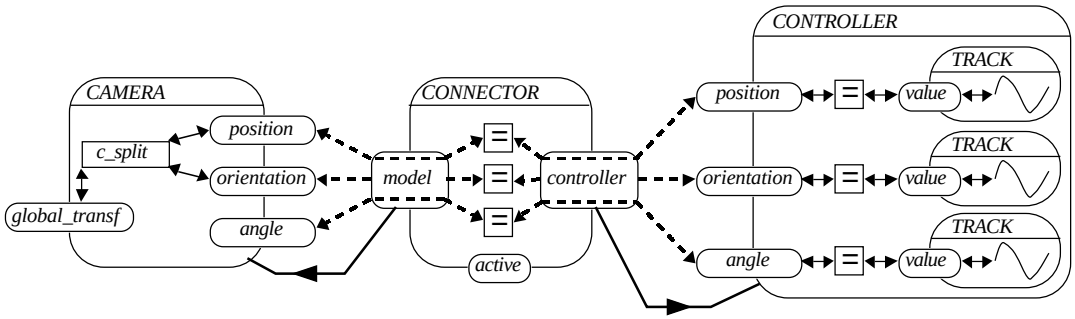


Figure 11. Connecting a controller to a camera.

5. Evaluation

In order to illustrate how the user can define an animation with *Virtual Studio*, we will explain the successive steps required to create a simple animation sequence.

The example scene is composed of three elements: a character, a light and a camera. An appropriate virtual tool has been connected to each scene element to be manipulated, and controller objects have been bound to each element in order to allow animation recording and playback. Figure 12 shows the initial configuration of the scene.

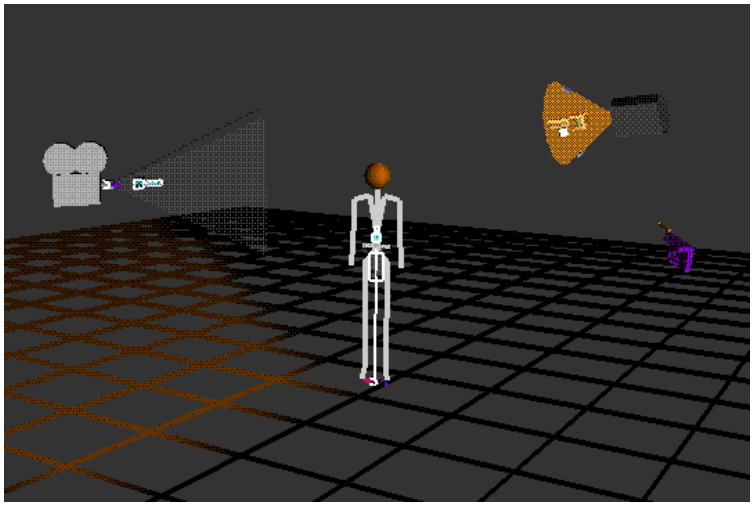


Figure 12. The scene to animate

The storyboard of the simple animation is as follows: the character will walk along a circular path and will stop near the starting point. Simultaneously, the light will follow a vertical path, while continuously aiming at the character's head. During the character's animation, the camera will stay at a fixed position, while always looking at the character's head. At the end of the character's motion, the camera will move towards the character's head and the light will fade out.

In order to define this sequence, the user performs the following steps:

- **light's animation**: the light tool is grabbed and the desired motion is interactively described in space and time by means of a 3D device (figures 13 and 14);
- **addition of a lookat constraint**: the light tool is constrained to permanently aim at the character's head by interactively adding a lookat constraint between the tool and the character's head. During playback, newly added behaviors participate in the animation and may override previously recorded motion. The light's orientation is now determined by the lookat constraint while its position is animated using the recorded information (figures 15 and 16);
- **character's animation**: a walk tool encapsulating a walking engine is connected to the character in order to provide a walking behavior to the model. The character is animated by grabbing the walk tool and by sketching the path the character has to follow. The light animation is automatically updated due to the lookat constraint (figure 17);
- **synchronization of light's and character's animations**: by connecting together the start binders and the end binders of the character's and light's tracks, both motions are made to perform in parallel. The animation of the light's orientation is again updated by the lookat constraint in response to timing changes (figure 18);
- **definition of the camera's motion**: the orientation binder of the camera tool is connected to the character's head; the position component of the path is subsequently sketched using the 3D device. At the end of the specification, the start binder of the camera's track is connected to the end binder of the character's track. The animation of the camera is now automatically scheduled after the animation of the character (figure 19);
- **definition of light's fade out**: Another way to synchronize animations is to interact with the environment during animation playback. Here, we define the animation of the light's intensity, a non geometrical attribute, using the mediation of a tool allowing the remote control of light's parameters. The resulting spline is automatically synchronized with the rest of the animation. During playback, information propagates from the animation tracks, through the light and down to the connected tools, which are forced to display the light state changes (figure 20).

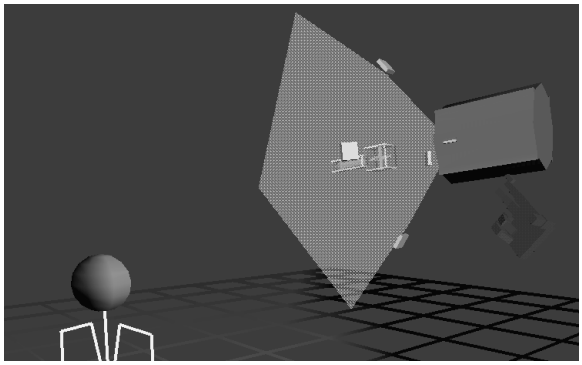


Figure 13. The animator grabs the tool connected to the light source to define the animation. The 3D path described with the device is recorded and re-represented as a B-spline using our incremental data reduction algorithm.

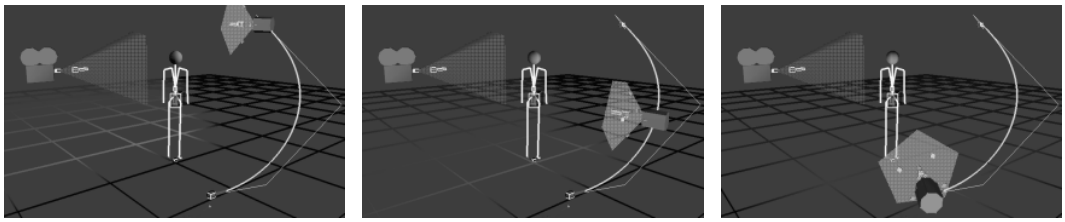


Figure 14. Three frames from the resulting sequence. The spline reproducing the interactively described motion and its control polygon are now shown.

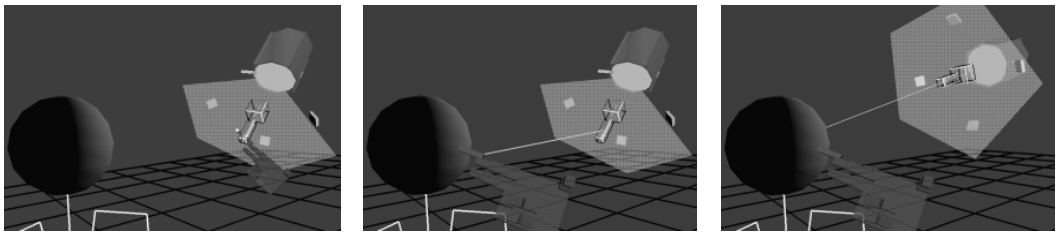


Figure 15. In order to have the light continuously pointing towards the character, a lookat constraint is interactively added by connecting the tool's orientation binder to the character's head. A line from the orientation binder to the head indicates the existence of the connection.

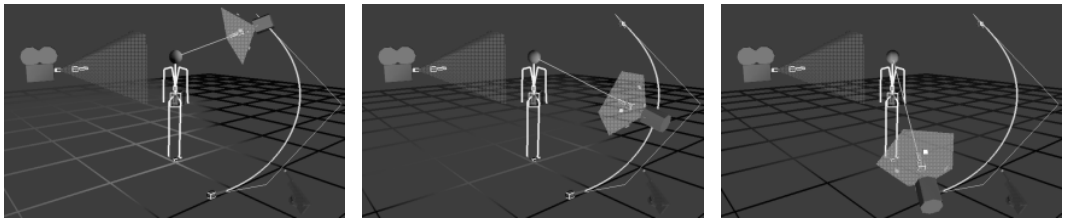


Figure 16. Due to the lookat constraint, the recorded animation of the light's orientation is now invalidated and the animation of the orientation component is fully determined by the constraint.

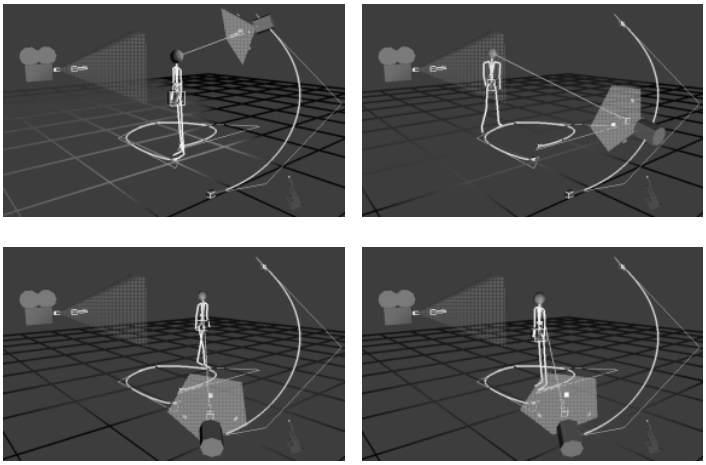


Figure 17. The animator sketches the path that the character has to follow. The behavior of the walk tool realizes the walking cycle animation along the path and filters the information during the interaction so as to translate the 3D motion specification to a 2D path on the floor.

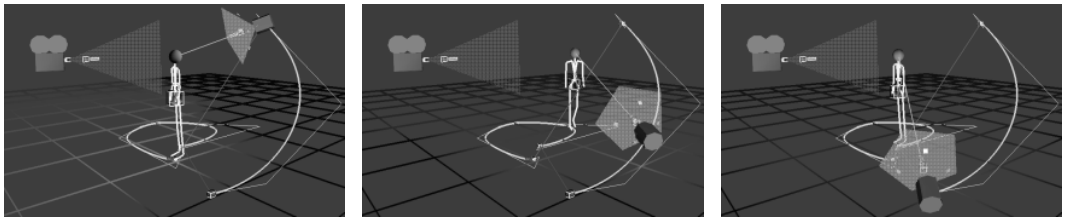


Figure 18. Synchronization constraints have been introduced interactively between the character's and light's tracks. Now both motions are performed in parallel.

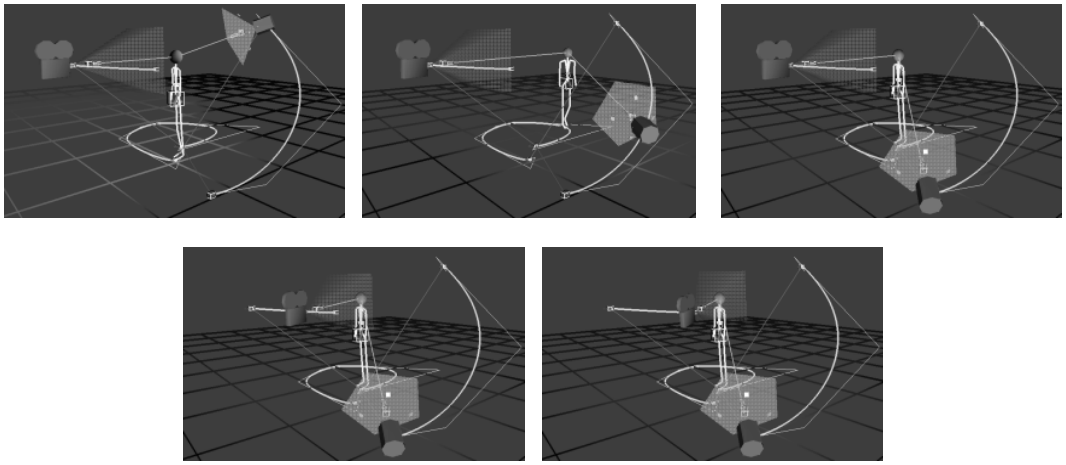


Figure 19. The animation has now been extended with the camera's motion. A synchronization constraint has been used to specify that the camera's motion must start when the animation of the character terminates.

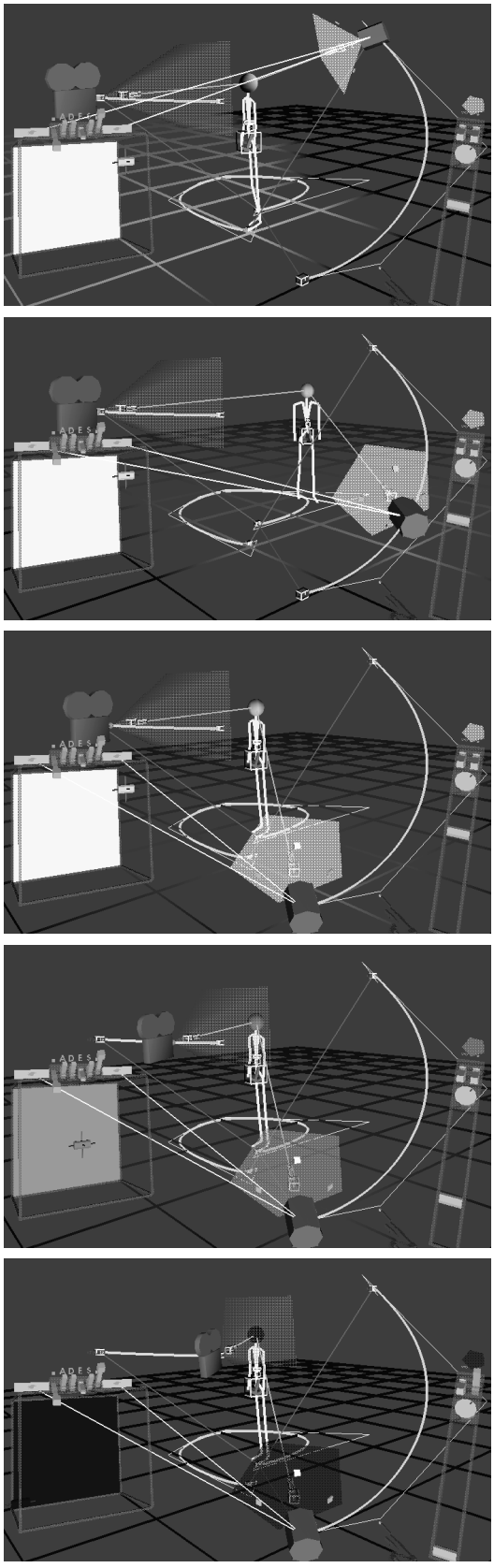


Figure 20. By interacting with the environment during play-back, animations containing complex synchronizations with other animations can be easily specified. In this example, a light fade-out animation, synchronized with the camera's motion, has been defined using a tool allowing the remote control of light parameters. During playback, all the connected tools display the light's state changes.

Thanks to the expressive power provided by 3D devices and virtual tools, the animator has been able to set up a first version of this sequence in less than five minutes. Since data reduction was applied when defining the evolution of all the continuous parameters, the resulting animation can now be edited using standard spline edition tools. The animator can therefore benefit at the same time from the expressiveness of straight-ahead animation specification and from the precision of key-frame techniques.

As shown by the lookat and the walking engine examples, virtual tools are not limited to simple editing tasks, but provide a mean to interactively add behavior to objects of the synthetic environment. Secondary animations can be automatically generated by virtual tools, simplifying that way the task of the animator that can concentrate his efforts to the definition of high level goals.

6. Implementation and Results

Virtual Studio is implemented in the object-oriented language *Eiffel* on *Silicon Graphics* workstations, and is currently composed of over 700 classes.

Complex environments composed of thousands of variables and constraints can be manipulated and animated at interactive speed. The environment used for creating the example animation of the previous section was composed of 5632 constraints and 13659 active variables at the moment figure 20 was generated. The example scene was composed of 3000 polygons illuminated by a spot light and a directional light. The redraw rate was 10 frames per second on a *Silicon Graphics Crimson VGX*. Despite the complexity of the constraint network, the redraw time was still the limiting factor: on average, 80% of the application time was spent in the rendering operation.

7. Conclusions and Future Work

We have demonstrated that 3D devices and virtual tools provide enough expressive power to rapidly prototype complex 3D animated environments. By recording the effects of manipulations and applying data reduction techniques, we are able to sketch animations and obtain editable representations of parameter evolution. We believe that *Virtual Studio* provides a good basis for integrating a large variety of interaction and animation techniques. We are currently extending it with tools to control the timing of animations and the deformations of objects.

Acknowledgments

We would like to thank Russell Turner for reviewing the content of this paper, Ronan Boulic for providing the walking engine, Marc Ledin and Gilles Van Ruymbeke for designing and building the *VB2* image splitter.

This research was conducted by the authors at the Swiss Federal Institute of Technology in Lausanne and was partly sponsored by *Le Fonds National Suisse pour la Recherche Scientifique*.

References

[1] Balaguer JF (1993) *Virtual Studio: Un système d'animation en environnement virtuel*. PhD Thesis. EPFL DI-LIG, Switzerland.

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

[3] Gobbetti E (1993) *Virtuality Builder II: Vers une architecture pour l'interaction avec des mondes synthétiques*. PhD Thesis. EPFL DI-LIG, Switzerland.

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

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

[6] Lasseter J (1987) Principles of Traditional Animation Applies to 3D Computer Animation. *Proc SIGGRAPH*: 35-44.

[7] Lyche T, Mörken K (1987) Knot Removal for Parametric B-spline Curves and Surfaces, *Computer Aided Geometric Design 4*: 217-230.

[8] Bryson S, Pausch R, Robinett W, van Dam A (1993), *Implementing Virtual Reality*, SIGGRAPH Course Notes 43.

[9] ACM SIGGRAPH (1993), *Character Motion Systems*, Course Notes 01.

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

[11] Forcade T (1993) Evaluating 3D on the High End. *Computer Graphics World*, Oct/Nov.

[12] Owen GS, Blystone RV, Miller VA, Mones-Hattal B, Morie J (1993) Facilitating Learning with Computer Graphics and Multimedia. Panel discussion. *Proc SIGGRAPH*: 383-384.

Jean-Francis Balaguer is a 3D interaction consultant for the Center for Advanced Studies, Research, and Development in Sardinia. He received his M.S. in computer science from the Institut National des Sciences Appliquées (INSA) in Lyon, France and his PhD in computer science from the Swiss Federal Institute of Technology in Lausanne. His research interests include 3D interaction, virtual reality, computer animation and object-oriented graphics. Jean-Francis Balaguer can be reached at CRS4, Via Sauro 10, 09123 Cagliari, Italy or on E-mail at balaguer@crs4.it.

Enrico Gobbetti is a 3D interaction consultant for the Center for Advanced Studies, Research, and Development in Sardinia. He received his M.S. and his Ph.D. in computer science from the Swiss Federal Institute of Technology in Lausanne. His research interests include object-oriented and constraint programming, 3D interaction, time-critical graphics, computer animation, and virtual reality. Enrico Gobbetti can be reached at CRS4, Via Sauro 10, 09123 Cagliari, Italy or on E-mail at gobbetti@crs4.it.