A New Interface Paradigm for Motion Capture Based Animation Systems

Fernando Wagner Serpa Vieira da Silva[1,2]
Luiz Velho[1]
Paulo Roma Cavalcanti[2]
Jonas de Miranda Gomes[1]

{nando,lvelho,roma,jonas}@visgraf.impa.br

[1]IMPA–Instituto de Matemática Pura e Aplicada
Estrada Dona Castorina, 110,
22460 Rio de Janeiro, RJ, Brazil

[2]LCG - Laboratório de Computação Gráfica, COPPE - Sistemas / UFRJ
Caixa Postal 68511,
21945-970, Rio de Janeiro, RJ, Brazil

Abstract. This paper proposes a new user interface paradigm for motion capture based animation systems, providing intuitive and efficient ways to visualize the main motion capture concepts and operations. A prototype system was built, implementing the proposed interface model and supported by a flexible architecture that is suitable to work with the motion capture methodology.

keywords: motion capture, animation systems, computer animation, graphic interfaces, GUI paradigm, motion control.

1 Introduction

The Motion Capture technique has already set its place in the future of computer animation. This technique provides tools for high-quality animation, even when real-time is required.

Initially, the main drawback of Motion Capture techniques was the lack of efficient ways to modify the captured motion, by adjusting or improving specific parts that needed to be changed, without having to repeat the entire acquisition process again.

Lately, however, several techniques were proposed to process captured data [9] [11] [12], providing tools for motion analysis, modification and reuse. This makes motion libraries more valuable for a wide class of animators.

Most of current animation systems offer the possibility of using motion captured data to generate animations, but several systems treat this technique as an "extra tool", or even as a simple plug-in. Therefore, an effective description of motion capture basic concepts is not provided.

An interesting and alternative approach would be the construction of an animation system that uses motion capture as the kernel of the entire animation process. This leads to an unlimited range of possibilities to many animators. As an example, this motion capture based system could be integrated with high-end motion capture hardware, thus creating a powerful environment of motion acquisition and processing.

In this work, we propose a new user interface paradigm for motion capture based animation systems, supported by an extensible architecture that incorporates the "state of art" in motion capture processing techniques, and allows the use of standard animation methods, such as keyframing or inverse kinematics, as powerful tools to improve the system's flexibility.

The main contribution of this paper is the introduction of an elegant way to describe, at the user interface level, the basic motion capture abstractions. We treat captured motion as a potentially ready animation, which can be modified by a set of tools embedded in the architecture. In that way, an interface description of motion operations and associated objects are defined.

A prototype system was built, implementing the concepts described in this work. This system was used to demonstrate the potential of the proposed interface paradigm.

Section 2 of this paper introduces the basic internal structures of the prototype system, together with a brief description of its architecture. Section 3 presents the proposed interface paradigm for motion capture based systems. Finally, conclusions are given in section 4.

2 System's Architecture and Internal Structures

In this section, we present a framework for the animation system. Also, a brief description of the architecture used in the implemented system is provided. This will give a better understanding of the approach used to build the proposed interface paradigm.

2.1 Basic Internal Structures

The fundamental structure used in the system is composed by two entities: an actor and motions.

The actor is treated as a skeleton. Its topology is represented by a graph formed by joints and links. Its geometry is represented by a set of connected limbs (Figure 1). This description is adequate to be used in a motion capture animation system, since it reflects the structure of an articulated figure. For data acquisition, markers are attached at the joints of a live performer (the real actor).

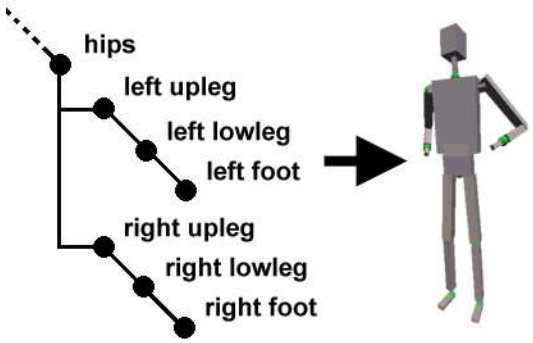

Figure 1: Topological and geometrical description of an actor used in the system.

At the programming level, the actor is represented using a modified version of Zeltzer's APJ (*Axis Position Joint*) structure [3], adapted to work with motion captured data.

Motions are best represented as curves in time (Figure 2).

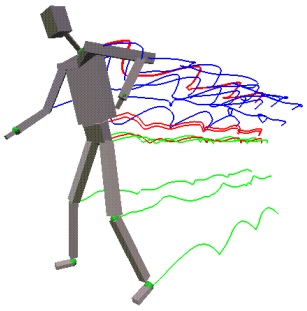

Figure 2: Examples of joint motion curves.

Normally, the captured data consists of marker's positional and / or angular variation, sampled by the capture hardware during the number of frames required to complete the actor's performance. This description is used in each degree of freedom (DOF) of the actor.

Internally, the interaction with the user is controlled by a dynamic data structure that represents the current "state" of all windows and main data structures existing in the system.

2.2 An Architecture Focused on Motion Capture

We developed a conceptual architecture, designed to work with the motion capture paradigm. It focuses on some technological aspects and embodies several techniques to deal with captured data, thus allowing the creation of reusable motion libraries by using a building block paradigm.

The framework of the architecture (Figure 3) is formed by three basic modules (*input*, *processing* and *output*), each one responsible for a specific set of tasks. The data structures used in the system were described in 2.1.

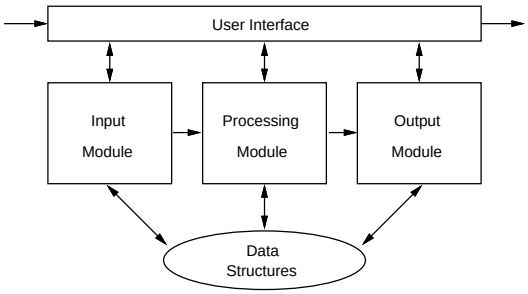

Figure 3: System's architecture.

All modules are supported by the conceptual interface that will be described in section 3. For a more detailed description of system's architecture, please refer to [4].

- **Input Module**

The input module focuses on problems concerning the interpretation and preprocessing of motion data. Skeleton definition files were created to establish relationships between the skeleton expected by the incoming data [1] and architecture's internal default skeleton definition. Also, geometrical algorithms for

[1] Depending on the motion data format.

3D Euler angle extraction[2], pre-filtering and hierarchical angle generation are provided in this module.

- **Processing Module**

The processing module comprises the set of tools for motion analysis, manipulation and reuse. Three basic motion operation types are defined: filtering; concatenation; and blending. Their objective is to provide efficient ways to modify the original captured data. With these tools, the user is able to generate new classes of motions, inheriting the aliveness and complexity typical of natural motion.

Filtering operations can be applied to the joint curves of a motion, to reduce noise or even modify specific components of the movement. In [9], Williams used a multiresolution filtering approach to decompose the motion in frequency bands, thus allowing modifications in a higher level of abstraction.

With concatenation operations, longer animations can be produced by combining several motions in sequence. Smooth transitions between the combined motions can be produced using algorithms based on blending of motion parameters [4]. Spacetime constraints [12] can also be used to generate seamless and dynamically plausible transitions, with excellent results.

Blending operations are normally used to combine special characteristics of different motions. In this case, the existence of tools for motion synchronization and reparametrization is very important to help in the blending process, ensuring a coherent result.

- **Output Module**

The main objective of this module is to provide ways to store the composition created by the user, thus maintaining and expanding the existing motion library.

A universal data format was defined, embodying the main characteristics of most motion data formats available nowadays. Consequently, the system can be used as a robust conversor to motion capture data formats.

- **User Interface**

The user interface used in the system is based on a visual representation of motion capture basic concepts, such as motions and operations.

We believe that it is also interesting to supply a non-graphical user communication, using a expression language that represents all the actions that would be done using the graphical description provided in the user interface.

[2]In the case of data with positional information only.

We are currently working on a expression language similar to that described in [12]. Using this language, the user will be able to generate complex animations with motion operations, using commands that will actually execute the callback functions supporting the user interface. These commands may be stored in a text file and can be reused or edited.

Control

A continuous loop (Figure 4) verifies the status of all interface objects and windows, reporting any changes to a special function that manages those changes that actually must be done due to user interaction.

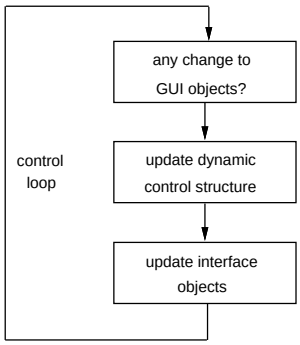

Figure 4: User interface control with a state checking loop.

3 A Graphic Interface for Motion Capture

As discussed before, our system is based on an architecture that treats the problem of dealing with captured motions. However, the functionality of its architecture would be shadowed by the conventional user interface paradigm used on most animation systems currently available.

Our goal is to describe the basic structures of the system's architecture in a concise way, providing powerful interface tools that will make it easier to execute motion editing operations. Moreover, this interface must be extensible, allowing the incorporation of new operations and techniques.

We noted that each captured motion is potentially a ready animation, and therefore must be treated accordingly. This prompted us to adopt an interface paradigm used in some post-production video workstations [10] as a starting point to our work.

In the following sub-sections, we will describe the main interface objects and concepts developed under the proposed methodology. All images presented in this section are actual screenshots of the prototype system, which implements the concepts described in this paper.

3.1 Actor

In our interface paradigm, the actor is visualized by means of its components: the skeleton topology and geometry. The skeleton structure is shown in a window as a graph composed of joints and links (Figure 5). This window displays information about each joint and the links between them. The interface allows joint selection and / or grouping.

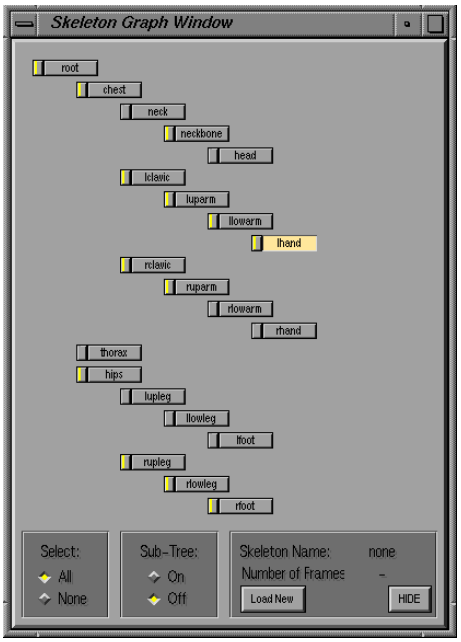

Figure 5: Skeleton graph window.

The representation of the actor can be visualized with different styles in the rendering window, as shown in Figure 6. The selected joint in the graph window appears in different color and size in the rendering window. This selection also affects other interface objects, as will be discussed later.

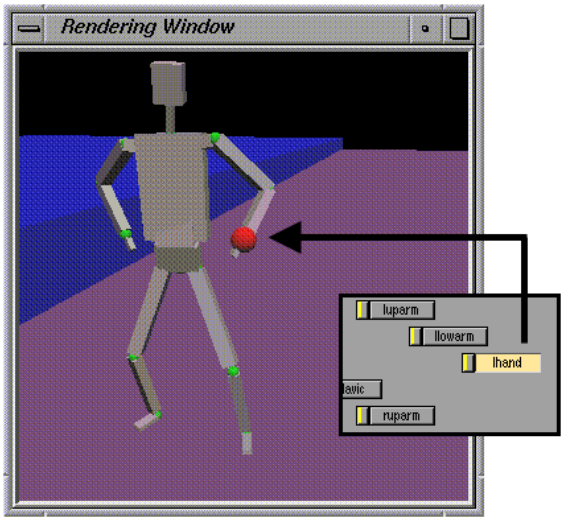

Figure 6: Joint selection (detail from skeleton graph window) and visual feedback in the rendering window.

3.2 Motions

In the post-production video interface model, video and sound sequences are visualized as horizontal bars, which can be grouped, positioned and combined in a timeline canvas, in order to produce the final composition.

In our paradigm, we treat a motion as a horizontal bar (Figure 7), whose width is determined by the number of frames of the captured motion. This bar also contains information about the motion name. Note the arrow marker at the right end of the motion bar, which indicates that motion resizing (reparametrization) is allowed.

Figure 7: Motion representation as a GUI object.

We will usually visualize the motion bar using a frame ruler associated with it. This gives a more accurate temporal perception of the motion.

- **The Motion ScratchPad**

In our system, we have created an interface object whose purpose is to act as a motion organizer, providing a global perception of all motions placed on it. We called this object the Motion ScratchPad (Figure 8).

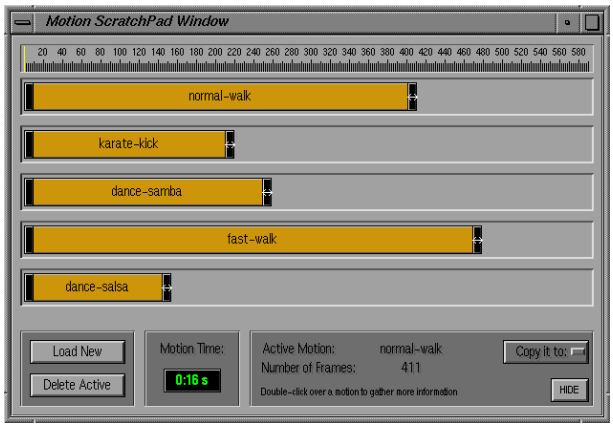

Figure 8: Motion ScratchPad - the motion organizer.

Using the ScratchPad, the user can choose several motions and organize them in the object canvas. All motions can be individually played or resized, and the user can drop them to the motion operations. The ScratchPad is actually the gateway between the input and the processing modules.

The ScratchPad is the container where motion fragments are stored, waiting to be used.

- **Joint Curves**

Joint curves are the basic components of motion. They are visually represented by an interface object that displays curve shape. It also provides numeric information about the different data channels at each frame (i.e., three position and orientation values, for each joint curve). The visual representation of a joint curve is shown in Figure 9.

With this representation, it is straightforward to implement several curve editing techniques [9] [11], allowing a precise and interactive control of the curve shape.

For each joint of the actor, there may be several joint curves (one for each DOF) attached to it. These curves are grouped in a interface object, the Joint Curves Window, that offers a global view of the curves, and has a direct connection with the skeleton graph window. When a joint is selected in the graph window, its curves are displayed and useful information is provided, as shown in Figure 10.

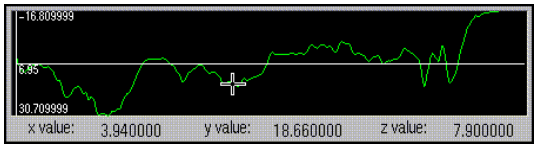

Figure 9: Visual representation of a joint curve.

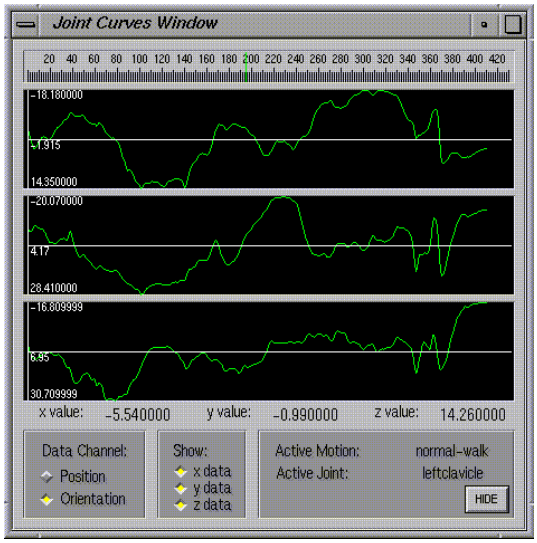

Figure 10: Joint Curves Window.

- **The Motion Window**

Finally, we decided to create an interface object that acts as a connection between motion and actor descriptions. The Motion Window (Figure 11) is accessed via a double-click in a motion bar, and allows the selection (or grouping) of specific joints of the actor.

The Motion Window is composed of several bars, each one representing an actor joint (detail in Figure 11). Visually, it looks like a zoom in the motion bar. This representation is intentional, since the motion is formed by the curves that are attached to actor's joints.

The Motion Window will proof its utility when used in conjunction with motion operations, allowing the application of an operation to a specific set of joints of the actor.

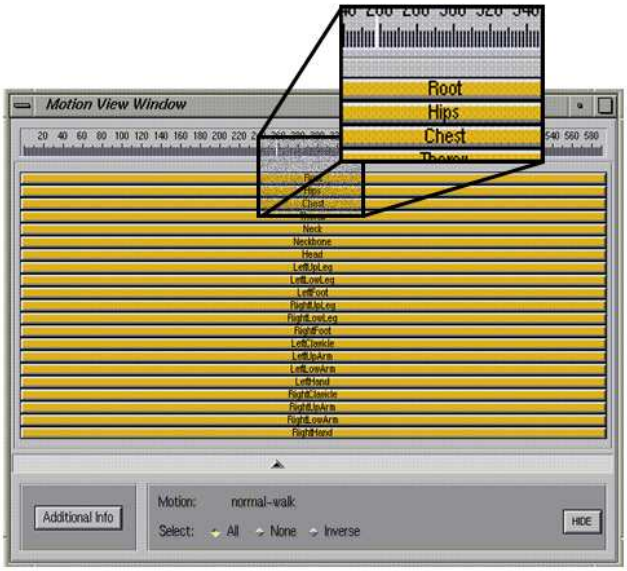

Figure 11: Motion Window - connecting actor and motion descriptions.

3.3 Motion Operations

Motion operations are the most important objects in the architecture. Therefore, they must be visualized in a way that makes the process intuitive to the user.

In our interface paradigm, each motion operation has its own window. When requested, additional objects are used to help in the process, providing a full control of the operation parameters.

All motion operation windows have a basic set of auxiliary objects: an interactive player slider and a frame ruler. These objects also follow the interface concepts described earlier.

- **Filtering**

The filtering operation (Figure 12) is represented in a window with tools which allow the selection of a specific region of the motion to be filtered (1), and with a list of the existing filters, which can be accessed by pressing 2 .

- **Concatenation**

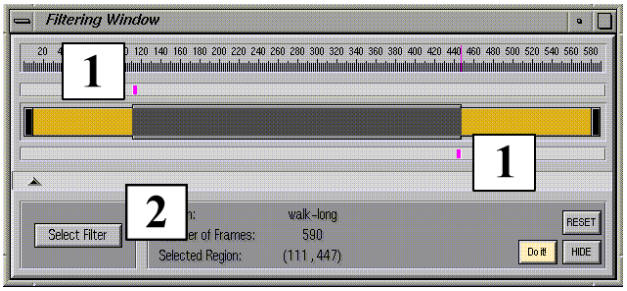

Figure 12: Filtering operation window.

The interface object that represents the concatenation operation was designed to provide a good visual perception of the composition as a whole.

The motions selected by the user in the ScratchPad are dropped into the Concatenation Window. Initially, they are positioned in such a way to perform a direct concatenation (i.e., without blending interval), as shown in Figure 13.

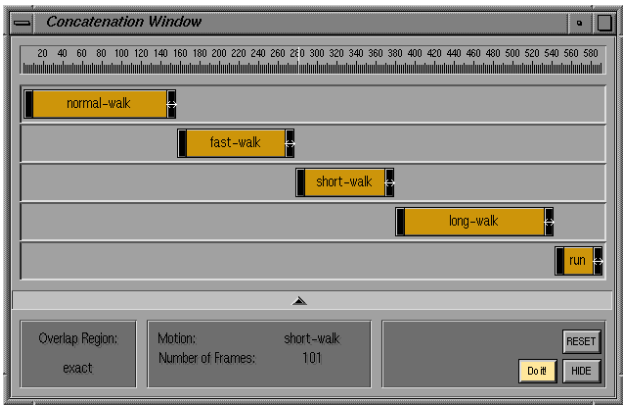

Figure 13: Concatenation operation window (initial motion arrangement).

Using the existing tools for motion positioning and reparametrization, the user can define blending intervals that will permit a smooth transition between the combined motions (in Figure 14, represented by the darker regions between motions).

However, some interface constraints were created to avoid undesirable results. These constraints ensure visual coherence, and guide the user in the operation process.

The default easy-in / easy-out blending parameters can be modified by double-clicking in the desired blending interval of the Concatenation Window.

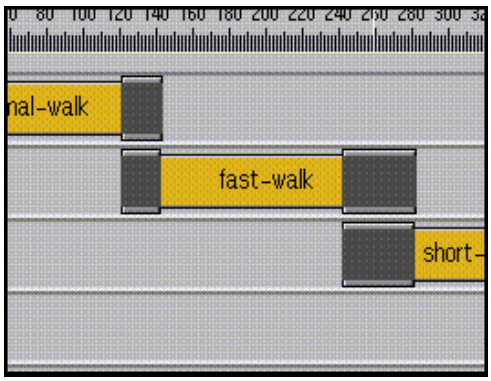

Figure 14: Blending intervals (detail from Concatenation Window).

- **Blending**

Blending is the last type of motion operation defined in the primary set of the prototype system.

As in the Concatenation Window, all motions selected by the user are placed in a canvas, providing a global view of the operation. For blending, however, new objects were introduced to assist in the specification.

These new objects are the time-markers, and their purpose is to synchronize key points in the combined motions. For example, when combining two different types of walk movements, it is desirable that the feet reach the ground at the same instant in both motions, otherwise strange results can be produced.

Figure 15 shows a snapshot of a blending operation between three different motions. The time-markers (detail from Figure 15) were used to establish a correspondence between the key steps in the combined motions.

This synchronization process will reparametrize the motions according to the position of the time-markers, matching their occurrence in time. To do that, algorithms based on timewarping are used. A good example can be found in [9].

3.4 Higher-Level Interface Objects

To complete the interface description, we present other objects that are important to create a powerful animation environment.

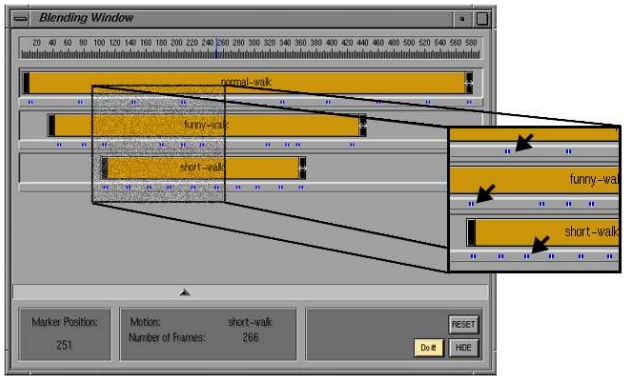

Figure 15: Blending window with time-markers (in detail).

- **Camera Controls**

Our system provides several tools for camera control. In Figure 16, we present the Camera Control Window, with some options that allow a precise control of various camera settings.

Among these options, probably the most useful ones are: the *follow mode* - which guides automatically the camera throughout the scene, following the root joint that drives the skeleton hierarchy; the *lock joint mode* - which points the camera target to the active joint, selected in the skeleton graph window; and the *circle camera* option, which allows an interactive circular movement of the camera over the active joint, while the animation is being played.

Moreover, these camera options can be mixed, thus giving a yet more precise control of the camera motion.

Additional controls for scene lighting are also provided with the system.

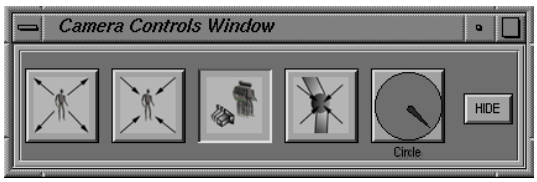

Figure 16: Camera controls. From top to bottom: Zoom in, Zoom out, Follow mode, Lock Joint mode, Circle camera

- **Playback Controls**

Following the post-production video interface, we developed a control panel, which allows interactive control of animations in our system (Figure 17).

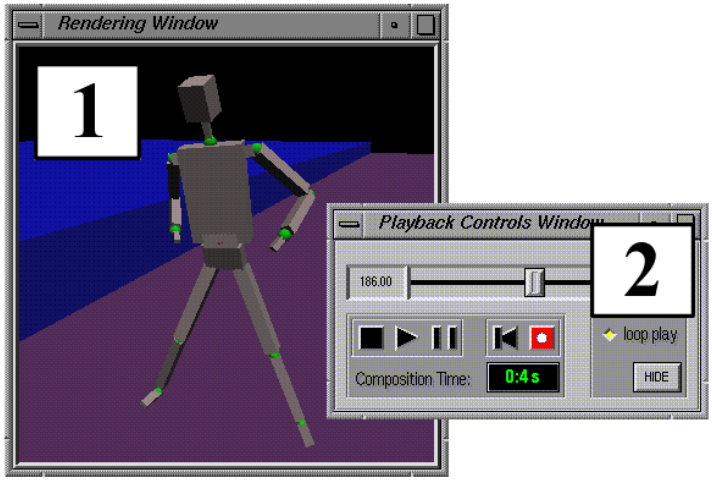

Figure 17: Playback and rendering windows.

The playback of motions or compositions produced by the user are executed in a dedicated OpenGL rendering window (in Figure 17, 1). The control panel is integrated with it (in Figure 17, 2), supplying a set of controls similar to those used in video recorders.

- **Objects for Other Animation Tools**

As discussed before, the architecture used in the system allows the integration of other animation techniques like keyframing, forward and inverse kinematics to help in the processing of captured motions.

In the current implementation, the system uses forward kinematics as an auxiliary tool to adjust the position of specific joints when necessary. The addition of keyframing and inverse kinematic tools is planned for future implementations.

3.5 Implementation Issues

The presented interface paradigm and prototype system were implemented in the programming language C, using a SGI Indigo 2 graphic workstation as the base platform. We employed OpenGL [14] for rendering and XForms [13] for

GUI generation. The advanced GUI objects were designed and implemented separately, and then added to the Forms library.

Due to OpenGL's rendering facilities and to the dynamic interface control used in the system, a real-time frame rate is achieved during the playback of animations (about 15 frames/sec in a SGI Indigo 2). The prototype system was also tested in the Linux and RISC 6000 platforms, also with good frame rates.

Figure 18 shows a snapshot of a typical system usage, with an arrangement that contain some of the previous described windows (Joint Curves Window, 1; Skeleton Graph Window, 2; Motion ScratchPad Window, 3; Concatenation Window, 4 and Playback and Rendering Window, 5).

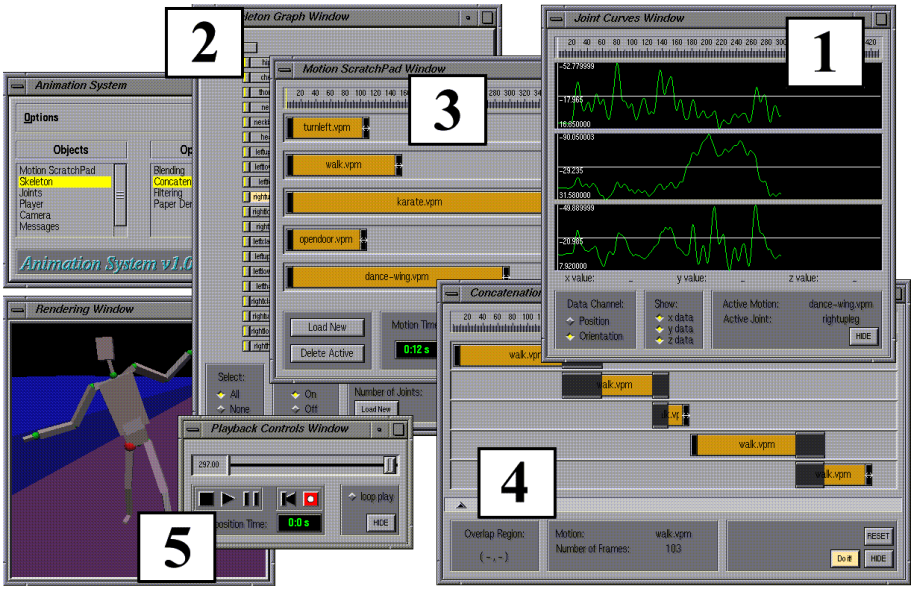

Figure 18: Snapshot of the prototype system.

4 Conclusions

In this paper, we have proposed a new user interface paradigm for motion capture based animation systems. A prototype system was built, employing the presented interface concepts and supported by a powerful architecture designed to work with the motion capture methodology.

The implemented system has proved to be easy and intuitive to use, with promis-

ing results that encourage us to improve it even more, with features like:

- implementation of other advanced motion operations ([9], [11], [12]), comparing their results and extracting conclusions and suggestions for improvements and / or new techniques. In this case, the system will serve as a test bed for new motion capture processing techniques.
- insertion of procedural [15] and behavioral [18] animation "plug-ins" in the system. In the first case, captured motions could act as a regent factor, guiding the procedural objects. In the second case, the behavioral functions could control the processing module, combining and modifying captured motions to improve the visual quality of the animations.
- combination of motion capture with sound. In this case, the time-markers could be used to synchronize the key moments in the motion with the temporal description of the sound.

5 Acknowledgements

The authors would like to thank Viewpoint Datalabs, Inc. and Biovision, Inc. for access to motion capture data, and to the Brazilian Council for Scientific and Technological development (CNPq) for the financial support. This research has been developed in the laboratory of VISGRAF project at IMPA and at LCG/UFRJ, as part of the Master programee of the first author. This project is sponsored by CNPq, FAPERJ, FINEP and IBM Brasil. Also thanks are due to the reviewers for their valuable comments.

6 References

[1] Ginsberg, C. M., *Human Body Motion as Input to an Animated Graphical Display*. Master Thesis, Massachusetts Institute of Technology, May 1983.

[2] Maxwell, D. R., *Graphical Marionette: A Modern day Pinocchio*. Master Thesis, Massachusetts Institute of Technology, June 1983.

[3] Zeltzer, D. and Sims, F., A Figure Editor and Gait Controller for Task Level Animation. In *Computer Graphics (SIGGRAPH'88), Course Notes*, no. 4, 164-181.

[4] Silva, F., Velho, L., Cavalcanti, P. and Gomes, J. M., An Architecture for Motion Capture Based Animation. (preprint), 1997.

[5] Dyer, S., Martin, J. and Zulauf, J., Motion Capture White Paper. Technical Report. Silicon Graphics, December 12, 1995.

[6] Character Motion Systems. In *Computer Graphics (SIGGRAPH'94)*, Course no. 9.

[7] MULDER, S., Human Movement Tracking Technology. Hand Centered Studies of Human Movement Project, Simon Fraser University. Technical Report 94-1, July 1994.

[8] O'ROURKE, J., *Computational Geometry in C.* Cambridge University Press, 1994.

[9] WILLIAMS, L. AND BRUDELIN, A., Motion Signal Processing. In *Computer Graphics (SIGGRAPH'95 Proceedings)*(August 1995), pp. 97-104.

[10] Turbo Cube / Video Cube - User's Guide. IMIX Company.

[11] WITKIN, A. AND POPOVIC, Z., Motion Warping. In *Computer Graphics (SIGGRAPH'95 Proceedings)*(August 1995), pp. 105-108.

[12] COHEN, M., ROSE, C., GUENTER, B. AND BODENHEIMER, B., Efficient Generation of Motion Transitions Using Spacetime Constraints. In *Computer Graphics (SIGGRAPH'96 Proceedings)*(August 1996), pp. 147-154.

[13] Xforms Home Page, *http://bragg.phys.uwm.edu/xform*

[14] NEIDER, J., DAVIS, T. AND WOO, M., *OpenGL Programming Guide - The Official Guide to Learning OpenGL, Release 1.* Addison-Wesley, 1993.

[15] PERLIN, K., Realtime Responsive Animation with Personality. In *IEEE Transactions on Visualization and Computer Graphics*, Vol 1, No.1, March 1995.

[16] WITKIN, A. AND KASS, M., Spacetime Constraints. In *Computer Graphics (SIGGRAPH'88 Proceedings)*(August 1988), pp. 159-168.

[17] TERZOPOULOS, D. ET AL., Artificial Fishes with Autonomous Locomotion, Perception, Behavior and Learning, in a Physical World. In *Proceedings of the Artificial Life IV Workshop*, MIT Press (1994).

[18] COSTA, M. AND FEIJÓ, B., An Architecture for Concurrent Reactive Agents in Real-Time Animation. In *Proceedings of SIBGRAPI'96, IX Brazilian Symposium of Computer Graphics and Image Processing*, pp. 281-288. 1996.