Production and Playback of Human Figure Motion for 3D Virtual Environments

John P. Granieri, Jonathan Crabtree, Norman I. Badler

Center for Human Modeling and Simulation
University of Pennsylvania
Philadelphia, PA 19104-6389, USA
granieri|crabtree|badler@graphics.cis.upenn.edu

Abstract

We describe a system for off-line production and real-time playback of motion for articulated human figures in 3D virtual environments. The key notions are (1) the logical storage of full-body motion in posture graphs, which provides a simple motion access method for playback, and (2) mapping the motions of higher DOF figures to lower DOF figures using slaving to provide human models at several levels of detail, both in geometry and articulation, for later playback. We present our system in the context of a simple problem: Animating human figures in a distributed simulation, using DIS protocols for communicating the human state information. We also discuss several related techniques for real-time animation of articulated figures in visual simulation.

1 Introduction

The ability to render realistic motion is an essential part of many virtual environment applications. Nowhere is this more true than in virtual worlds containing simulated humans. Whether these human figures represent the users' virtual personae (avatars) or computer-controlled characters, people's innate sensitivity as to what looks "natural" with respect to human motion demands, at the very least, that moving characters be updated with each new frame that the image generator produces.

We first discuss a topical problem which requires the real-time rendering of realistic human motion, and then describe our system for authoring the motion off-line, and playing back that motion in real time. We also address some of the issues in real-time image generation of highly-articulated figures, as well as compare several other methods used for real-time animation.

2 Human motion in DIS

The problem we are interested in is generating and displaying motion for human figures, in particular soldiers, in distributed virtual environments. Parts of the general problem and the need for representing simulated soldiers (referred to as Dismounted Infantry, or DIs), are covered in [15, 5]. Although primarily driven by military requirements today, the general technologies for projecting real humans into, and representing simulated humans within, virtual environments, should be widely applicable in industry, entertainment and commerce in the near future.

The Distributed Interactive Simulation (DIS) [7] protocol is used for defining and communicating human state information in the distributed virtual environment. The DIS protocol, at least the part relating to human entities, is in its early stages of development, and fairly limited in what it can specify about a human figure [11], but is a good baseline to start with. Our purpose here is not to engage in a discussion of the intricacies (nor worth) of the DIS protocol, but merely to use it as an example of a distributed simulation protocol which can communicate state information on a simulated human entity between simulation nodes in a network.

The information representing a human entity is currently defined by several discrete enumerations in the appearance field of an Entity State Protocol Data Unit (PDU) in the DIS protocol [8]. The relevant information we are interested in from the Entity State PDU is shown in Fig. 1. The human is always in one of the four postures, along with a weapon state. The heading defines the forward direction. Although there are enumerations for walking and crawling, we use combinations, such as (posture=*standing*)+(velocity>0) to be equivalent to walking or running. Although the protocol allows for up to three weapons of different types on a soldier, we only modeled one, a rifle.

If the human can be in any of n possible postures, there are potentially n^2 transitions between the postures. Rather than create n^2 posture transitions, we encode the postures and transitions into a *posture graph* [1]. The graph defines the motion path to traverse to move the human figure from any one posture to another. These graphs are directed and may include cycles. It also provides the logical structure for the run-time motion database.

When the velocity of the human is zero, the possible transitions between *static* (for lack of a better term) postures are encoded in the posture graph of Fig. 2. A few of the actual postures are shown in Fig. 3. In

0-8186-7084-3/95 $04.00 © 1995 IEEE

Field	Value	Units
Posture	*Standing*	
	Kneeling	
	Prone	
	Dead	n/a
Weapon	*Deployed*	
	Firing	n/a
Position	P_x, P_y, P_z	meters
Velocity	V_x, V_y, V_z	meters/second
Heading	*theta*	degrees

Figure 1: Essential human state information in a DIS Entity State PDU

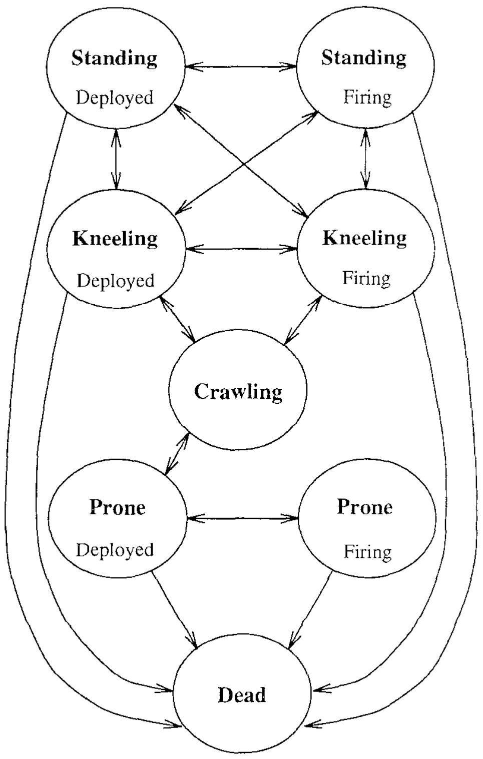

Figure 2: The *static* posture graph

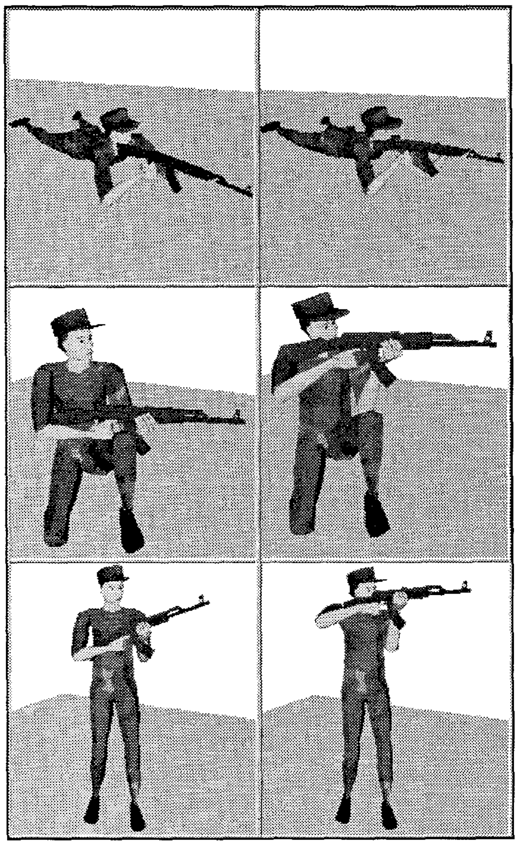

Figure 3: Some of the static postures a soldier can take in DIS

the posture graph, the nodes represent static postures, and the directed arcs represent the animated full-body transitions, or movements, from posture to posture. Each arc has an associated time for traversal, which is used to find the shortest path, in time, if more than one path exists between a starting posture and a goal posture.

When the velocity of the figure is non-zero, the possible transitions between locomotion postures are shown in the posture graph of Fig. 4. In this graph, the nodes are static postures, but the figure would never be in the posture for more than one frame.

The system we built consists of two distinct parts: 1) the off-line motion data generator, and 2) the on-line real-time playback mechanism, running in a high-performance IRIS Performer-based [12] image generator application.

3 Off-line motion production

Motion production involves three steps: 1) creating postures and motion for each node and arc in a posture graph, for one human model, 2) mapping the resulting motion onto human models with lower degrees-of-freedom (DOF) and lower resolution geometry, and

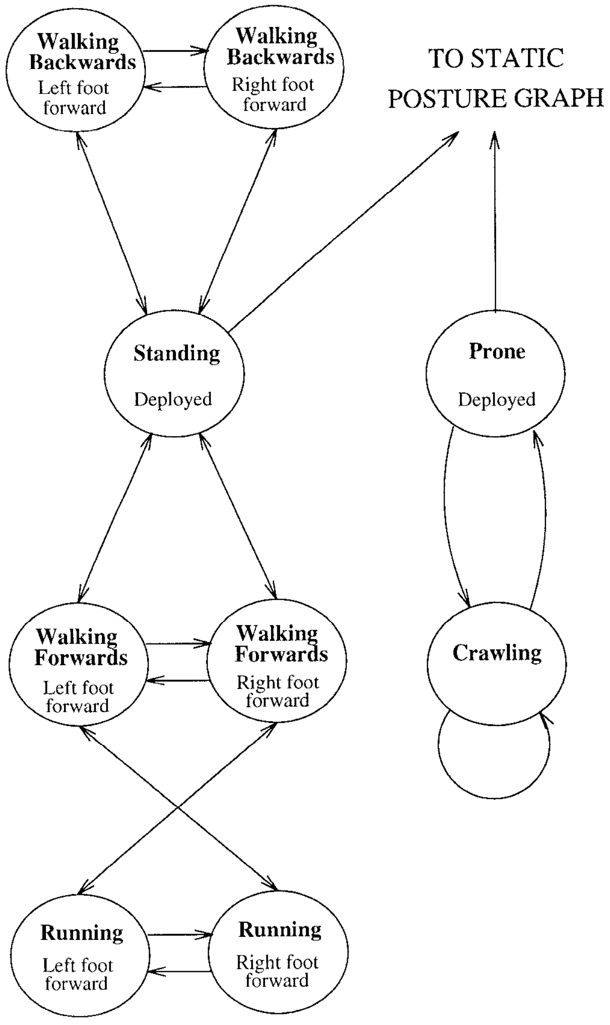

Figure 4: The *locomotion* posture graph

finally 3) recording the results and storing in a format for easy retrieval during playback.

3.1 Authoring the motion

The first step in producing motion for real-time playback is to create postures representing the nodes in the posture graphs, as well as the corresponding motions between them, represented as the directed arcs in the graphs. We used a slightly modified version of the *Jack* human modeling and animation system [2] for this purpose. *Jack* provides a nice constraint-based, goal-driven system (relying heavily on inverse-kinematics and primitive "behavioral" controls) for animating human figures, as well as facilities for organizing motions for general posture interpolation [1]. It is important to note that the posture graphs presented in this paper differ from the *posture transition graphs* presented in [1]. In the latter, the posture transition graphs are used to organize motion primitives for general posture interpolation with collision avoidance. In the former application (this paper) the posture graphs are a logical mechanism for organizing a database of pre-recorded motion, and determining motion sequences as paths between nodes of the graph. An underlying assumption of the posture graphs is that the articulated human figure's motion is continuous, and therefore can be organized into a connected graph.

Each directed transition in the static posture graph typically was produced from 10 to 15 motion primitives (e.g. `move_arm`, `bend_torso`, etc). Many of the directed motions from a posture node A to a posture node B are simply run in reverse to get the corresponding motion from posture B to posture A. In several cases, the reverse motion was scripted explicitly for more natural results.

The human figure can also move (either forwards or backwards, depending on the difference between the heading and the direction of the velocity vector) by either locomoting (if posture is standing) or crawling (if posture is prone). The locomotion posture graph transitions of Fig. 4 were generated by Hyeongseok Ko's walking system [9]. Six strides for each type of walking transition were generated (forward walking, backward walking, running): left and right starting steps, left and right ending steps, and left and right cyclic steps. The crawling animation was generated manually, and is based on two animations - one that goes from the prone posture to the cyclic state, and one complete cyclic motion. Note that only straight line locomotion of fixed stride is modeled. We are currently working on extending the system to handle variable stride length and curved path locomotion, as possible in the system of [9].

3.2 Slaving

The second step in the production process is concerned with preparing the motion for the real-time playback system. We wish to have tens, and potentially hundreds, of simulated humans in a virtual environment. This neccesitates having multiple level-of-detail (LOD) models, where the higher resolution models can be rendered when close to the viewpoint, and lower resolution models can be used when farther

	human–1	human–2	human–3
polygons	2400	500	120
rigid segments	69	19	12
joints	73	17	11
DOFs	134	50	21
motion	60Hz	30Hz	15Hz

Figure 5: The different levels of detail for the human models

away. We reduce the level of detail in the geometry and articulation by creating lower-resolution (both in geometry and articulation) human figures, with the characteristics listed in the table of Fig. 5.

All the motions and postures of the first step are authored on a (relatively) high resolution human body model which includes fully articulated hands and spine. This model is referred to as "human–1" in the above table. We manually created the two lower-resolution models, human–2 and human–3. Because of the difference in internal joint structure between human–1 and the lower LOD models, their motions cannot be controlled by the available human control routines in *Jack* (which all make assumptions about the structure of the human figure - they assume a structure similar to human–1). Instead of controlling their motion directly, we use the motion scripts generated in the first step to control the motion of a human–1, and then map the motion onto the lower resolution human–2 and human–3. We call this process *slaving*, because the high resolution figure acts as the *master*, and the low resolution figure acts as the *slave*.

Even though the different human models have different internal joint structures and segment shapes, their gross dimensions (e.g., length of arms, torso, etc.) are similar. The slaving process consists of interpolating the motions for the full human figure, generating all the in-between frames, and simultaneously having a lower LOD human model (human–2 or human–3) slaved, and then saving the in-between frames for the soldier. We will describe the process used for slaving from human–1 to human–2; the case with human–3 is similar.

For each frame of an animation, we first compute the position and joint angles for human–1. Then, an approximation of the joint angles for human–2 are computed. This is straightforward, as certain joints are the same (the elbow, for example, is only one DOF on both human models), and others can be approximated by linear combinations (for example, the 35 DOFs of the spine on human–1 can be summed and mapped directly onto the 7 DOF torso of human–2). This gives a good first approximation of the posture mapping, and provides an initial configuration for the final mapping. For the resulting motion of human–2 to look correct, we need to have certain landmark sites of the two bodies match exactly (the hands must be on the rifle). The final mapping step involves solving a set of constraints (point-to-point and orientation), to bring the key landmark sites into alignment. The

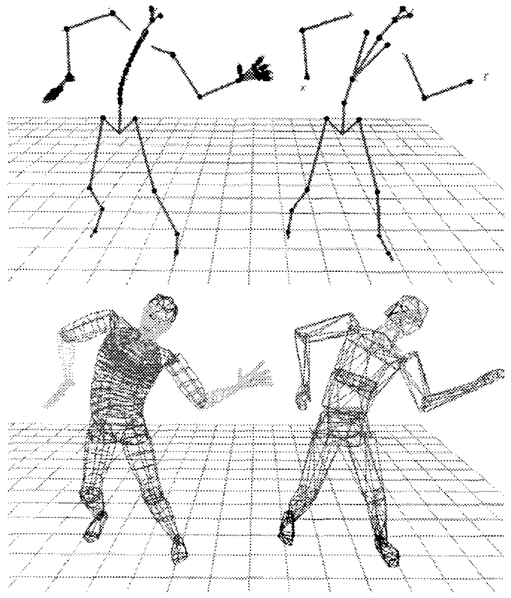
Figure 6: human–1 and human–2 models during slaving. human–1 is the master. Upper window is the skeletal articulation. Models are offset for illustrative purposes.

constraints are solved using an iterative inverse kinematics routine [17] to move the body parts into alignment.

Because of differences in geometry between the master and slave, in general we cannot expect all the landmark sites to match exactly. For the problem domain of this paper, animating the DIS protocol, the hands are always holding a rifle, so matching the hand positions accurately from the master is very important (otherwise the slave's hands may penetrate the rifle). Using a priority scheme in evaluating constraints, we assign higher priority to the hand-matching constraints than others, to account for this fact. If the slaving procedure cannot fit the master and slave within a certain tolerance, it will generate a warning for the animator.

3.3 Recording

The final step in the motion production process is to record the resulting motions of the human figures. The recorded motion for one transition is referred to as a *channel set* (where each joint or figure position is referred to as a *channel*; the channel is indexed by time). For each LOD human figure, a homogeneous transform is recorded, representing figure position relative to a fixed point, and for each joint, the joint angles are recorded (one angle per DOF). Also for joints, the composite joint transform is pre-computed and stored as a 4x4 matrix (which can be plugged directly into the parenting hierarchy of the visual database of the run-time system). Each channel set has an associated

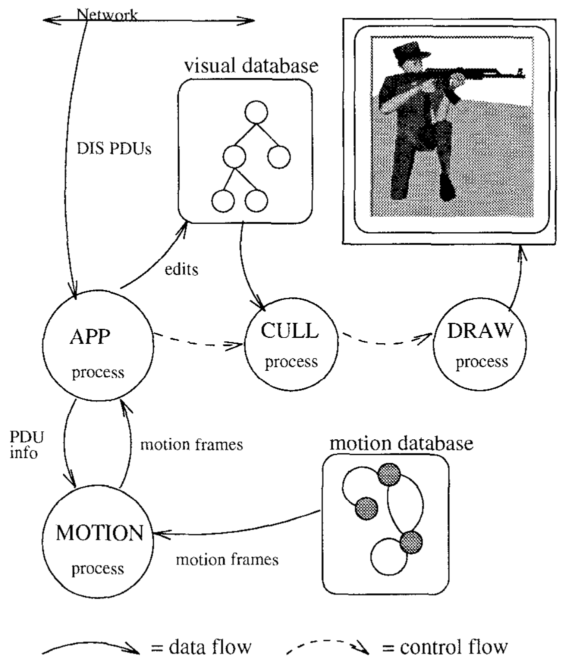

Figure 7: Overview of multi-processing framework for run-time system.

transition time. The channels of human-1 are interpolated and stored at 60Hz, human-2 at 30Hz, and human-3 at 15Hz. These rates correspond to the motion sampling during playback (see below).

4 Real-time motion playback

The real-time playback functions are packaged as a single linkable library, intended to be embedded in a host IRIS Performer-based visual simulation application. The library loads the posture graphs shown in Fig. 3 and 4, as well as the associated channel set motion files. Only one set of motions are loaded, and shared amongst any number of soldier figures being managed by the library. The articulated soldier figures themselves are loaded into the Performer run-time visual database. The library runs as a separate process, the MOTION process, serving motion data to the APP process (the APP, CULL and DRAW process are defined in the Performer multiprocessing framework). See Fig. 7 for a schematic overview of the runtime system.

An update function in the APP process is provided which maps joint angle values into the joint transforms of the soldier figures in the Performer visual database.

The APP process sends requests to the MOTION process, and receives joint angle packets back from the library. The content of the request to the library is simply the state information extracted from a DIS Entity State PDU, as shown in Figure 1. A simple control function translates these requests into playbacks of channel sets (the traversal of arcs of the posture graphs).

In the case of a static posture change (a motion from the static posture graph of Figure 2), the system will find the shortest path (as defined by traversal time) between the current and goal postures in the graph, and execute the sequence of transitions. For example, if the posture graph is currently at Standing Deployed, and Prone Firing is requested, it will transition from Stand Deployed to Crawl to Prone Deployed, and finally to Prone Firing.

The same shortest-path traversal method is used for executing posture changes in the locomotion posture graph of Fig. 4. It is important to realize that the only difference between the "static" and "locomotion" posture graphs is conceptual; the data structures involved are identical, and the distinction merely has to do with the conditions under which posture transitions are made. A posture change is made with a node of the static graph as a destination only upon receipt of a DIS Entity State PDU indicating that the agent is in such a posture. In the absence of further information, the agent **remains** in that posture. Conversely, when a posture change is made with a node of the locomotion graph as the destination, something that will occur if a PDU indicates the agent now has a non-zero speed, the agent does **not** remain in that posture once it is reached; absence of further information in this case means that the agent's speed is still nonzero, and hence the agent must take another step, or crawl another meter forwards, or whatever is appropriate for the current mode of locomotion. This continued motion requires that another posture change be made immediately.

One may think of labeling the transition arcs between posture graph nodes with conditions, as in a finite state machine. For instance, the transition from Standing Deployed to Walking Forwards (left foot forward) is taken whenever the agent's speed becomes non-zero and the agent's heading vector agrees with the velocity vector. On the other hand, if the vectors are not pointing in approximately the same direction, a transition is instead made to one of the Walking Backwards states. While the agent's speed remains nonzero (as it is assumed to in the absence of PDU updates), the system continually makes transitions back and forth between, for example, the Walking Forwards (left foot forward) and Walking Forwards (right foot forward) nodes. This cycle of transitions creates a smooth walking motion by concatenating successive left and right steps. Note that since we currently have no cycles of more than two nodes, finding the shortest path between postures in a cycle is a trivial matter!

Crawling is handled similarly, though it is a simpler case; there is no need for separate "left foot forward" and "right foot forward" states.

The system samples all the pre-recorded motion using elapsed time, so it is guaranteed to always play back in real time. For a 2 second posture transition recorded at 60fps, and a current frame rate of the image generator of 20fps, the playback system would play frames $0, 3, 6, \ldots, 120$. It recomputes the elapsed transition time on every frame, in case the frame rate of the image generator is not uniform.

The motion frame update packets sent from the

MOTION process back to the APP process are packaged to only include those joint angles which have changed from the last update. This is one way we can minimize joint angle updates, and take advantage of frame-to-frame coherence in the stored motions [1]. A full update (all joint angles and figure positions) is about 400 bytes.

4.1 Motion level-of-detail

It is recognized that maintaining a constant frame rate is essential to the believability of a simulation, even if it means accepting an update speed bounded by the most complex scene to be rendered. Automatic geometric level-of-detail selection, such as that supported by the IRIS Performer toolkit, is a well-known technique for dynamically responding to graphics load by selecting the version of a model most appropriate to the current viewing context [4, 6, 14].

The LOD selection within the visual database seeks to minimize polygon flow to the rendering pipeline (both in the software CULL and DRAW components of the software pipeline, as well as to the transformation engines within the hardware pipeline).

Given our representation, which enforces the separation of geometry and motion, it is possible to expand level of detail selection into the temporal domain, through *motion level-of-detail* selection. In addition to reducing polygon flow, via selecting lower LOD geometric models, we also are selecting lower LOD articulation models, with fewer articulation matrices, as well as sampling motion at lower frequencies. This reduces the flow of motion updates to the articulation matrices in the visual database. The models we are using are listed in Fig. 3.2.

In the playback system, we simultaneously transition to a different geometric representation with a simpler articulation structure, and switch between stored motions for each articulation model. We gain performance in the image generator, while consuming more run-time storage space for the motions. Our metric for LOD selection is simply the distance to the virtual camera. This appears to work satisfactorily for our current application domain, but further evaluation of the technique, as well as more sophisticated selection metrics (e.g. the metrics described in [6, 4]) need to be explored.

5 Example implementations and uses

The real-time playback system is currently being used in two DIS-based applications to create motion for simulated soldiers in virtual environments.

The Team Tactical Engagement Simulator [15] projects one or more soldiers into a virtual environment, where they may engage hostile forces and practice coordinated team activities. See Fig. 8 for a sample view into the training environment. The soldier stands in front of a large projection screen, which is his view into the environment. He has a sensor on his head and one on his weapon. He locomotes through the environment by stepping on a resistive pad and controls direction of movement and field of gaze by turning his head. The soldier may also move off the movement pad, and the view frustum is updated accordingly based on his eye position (head-coupled display). This allows the soldier, for example, to crouch down to see under a parked vehicle, or to peek around the corner of a building while still affording himself the protection of the building. TTES also creates the necessary DIS Entity State PDUs to represent the real soldier (mapping from sensor values into the small set of postures in the Entity State PDU), and sends them out over the net to other TTES stations that are participating in the exercise.

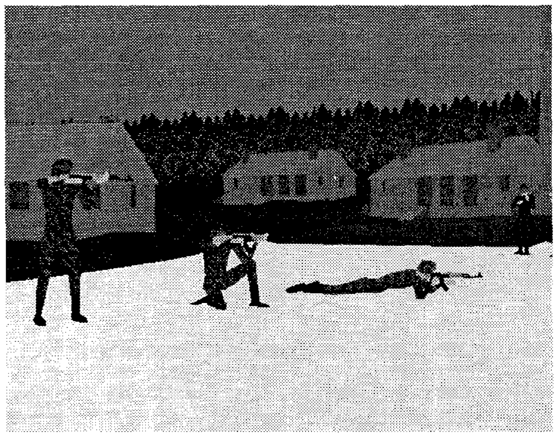
Figure 8: A View of Battle Town with several soldiers in different postures

The playback system is also used in a version of the NPSNET-IV [5] system, for generating motion of SIMNET- and DIS-controlled soldier entities.

Motion level-of-detail selection is of particular relevance to the example projects described above, because in the situation where a hostile agent enters the field of view of a soldier (one of the real human participants) and brings his weapon into the deployed position, the hostile's actions will probably be noted only in the participant's peripheral vision. It is well-known that humans can detect the presence of motion in their peripheral vision very easily, but that resolution of detail is very low. When head-tracking data is available or a head-mounted display is in use it is possible to designate areas of the viewing frustum as peripheral and reduce geometric and motion detail accordingly (not just based on linear distance to the camera, but angular offsets also). In the TTES environment this "focus of attention" information can be obtained from the aim of the real soldier's rifle when it is in the raised position, as the real soldier will almost certainly be sighting in this situation.

[1] An initial implementation of the playback library was run as an independent process, on another machine, from the host image generator, and joint angle packets were sent over TCP/IP stream sockets, hence the desire to minimize net traffic.

6 Comparison of production/playback methods

One of the most obvious criteria for evaluating a given motion representation is size; there is a clear progression in the methods used to animate humans (or any entity whose geometric representation varies over time) based on the amount of space required to store a given motion. We look at three methods.

The first method, requiring the most storage, involves generating and rendering the movements of characters in an off-line fashion. Frame-by-frame, a sequence of two-dimensional snapshots is captured and saved for later playback. The image generator then displays the bit-mapped frames in sequence, possibly as texture maps on simple rectangular polygons. Hardware support for texture mapping and alpha blending (for transparent background areas in the texture bitmaps) make this an attractive and fast playback scheme. Furthermore, mip-mapping takes care of level-of-detail management that must be programmed explicitly in other representations. Since the stored images are two-dimensional, it is frequently the case that artists will draw each frame by hand. In fact, this is precisely the approach utilized in most video games for many years. It is clear that very little computation is required at run-time, and that altering the motions incurs a high cost and cannot be done in real time. In fact, modifying almost any parameter except playback speed must be done off-line, and even playback speed adjustments are limited by the recording frequency. However, one real problem with using two-dimensional recording for playback in a three-dimensional scene is that non-symmetric characters will require the generation of several or many sets of frames, one for each possible viewing angle, increasing storage requirements still further. The authors of the popular game DOOM [13] record eight views of each animated character (for each frame) by digitizing pictures of movable models, and at run time the appropriate frames for the current viewing angle (approximately) are pasted onto a polygon. These eight views give a limited number of realistic viewing angles; it is impossible, for instance, to view a DOOM creature from directly above or below. Interestingly enough, an article on plans for a follow-up to DOOM reveals that the authors intend to switch to one of the two remaining representations we describe here:

> Unlike the previous games, the graphic representation of characters will be polygon models with very coarse texture mapping. This will make it hard to emulate natural locomotion, so they'll stay away from creating too many biped characters.[16]

Making the move to the second method involves a relatively slight conceptual change, namely taking 3-dimensional snapshots instead of 2-dimensional snapshots. This means storing each frame of a figure's motion as a full three-dimensional model. Doing so obviates the need for multiple data sets corresponding to multiple viewing positions and shifts slightly more of the computational burden over to the image generator. Instead of drawing pixels on a polygon the run-time system sends three-dimensional polygonal information to the graphics subsystem. It is still an inflexible approach because the figures are stored as solid "lumps" of geometry (albeit textured), from which it is extremely difficult, if not impossible, to extract the articulated parts of which the original model is comprised. Modifications must still be effected off-line, although rendering is done in real time. This is essentially the approach used by the SIMNET image generators to display soldiers on a simulated battlefield [3].

The final method is the one implemented by the system described in this paper, in which we record not the **results** of the motions, but the motions themselves. We store a single **articulated** three-dimensional model of each agent, and from frame to frame record only the joint angles between articulated segments. Modern rendering toolkits such as the IRIS Performer system used in this project increasingly allow support for storing coordinate transformations within a visual database, with relatively little cost associated with updating the transformation matrices in real time. As a result of adopting this approach, storage space is reduced and it is far easier to accurately perform interpolation between key frames because articulation information is not lost during motion recording. It also allows for virtual agents with some motions replayed strictly "as-is" and some motions which may be modified or generated entirely in real time. For instance, the slight changes in shoulder and elbow joint orientation required to alter the aim of a weapon held by a virtual soldier could be generated on demand.

We believe that the smallest representation presented in our size hierarchy, the third method, actually retains the **most** useful information and affords the most flexibility, while placing an acceptable amount of additional computational burden on the run-time display system.

7 Extensions & future work

We are currently exploring several extensions to the techniques described above, to add more expressive power to the tool bag of the real-time animator.

Key-framing and interpolation The use of the pre-recorded motions in the above posture graphs trades time for space. We do not compute joint angles on the fly, but merely sample stored motions. As the motions become more complex, it becomes very time-consuming to produce all the motions in the off-line phase, so we only produce key frames in a transition, every 5 to 10 frames, and then use simple interpolations to generate the inbetweens during real-time playback. This technique can't be extended much beyond that, as full-body human motion does not interpolate well beyond that many frames. This also reduces the amount of stored motions by a factor proportional to the spacing of the key frames, but increases computation time when a playback frame lands between two key frames.

Partitioning full-body motion In the posture graphs described previously, each motion transition included all the joint angles for the whole body. A technique to reduce motion storage, while increasing playback flexibility, is to partition the body into several regions, and record motion independently for each region. For example, the lower body can be treated separately during locomotion, and the upper body can have a variety of different animations played on it. Also, to support the mapping of motion from partially sensed real humans (i.e. sensors on the hands) onto the animated human figures, we want to animate the lower body and torso separately, then place the hands and arms using a fast inverse kinematics solution.

Articulation level-of-detail The various LOD models we used for the human figures were all built manually. Techniques for synthesizing lower LOD geometric models are known, but they don't apply to building lower articulation LOD models. Some techniques for automatically synthesizing the lower articulation skeletal models, given a high resolution skeleton and a set of motions to render, would be very useful.

Other dynamic properties A limitation is currently imposed by the fact that the segments of our articulated figures must be rigid. However, this is more an implementation detail than a conceptual problem, since with sufficient computational power in the run-time system real-time segment deformation will become possible. In general it seems likely that the limiting factor in visual simulation systems will continue to be the speed at which the graphics subsystem can actually render geometry. The adoption of coarse-grained multiprocessing techniques [12] will allow such operations as rigid or elastic body deformations to be carried out in parallel as another part of the rendering pipeline. The bottom line is that greater realism in VR environments will not be obtained by pouring off-line CPU time and run-time space into rendering and recording characters in exacting detail; the visual effect of even the most perfectly animated figure is significantly reduced once the viewer recognizes that its movements are **exactly** the same each and every time it does something. We seek to capitalize on the intrinsically dynamic nature of interacting with and in a virtual world by recording less information and allowing motions to be modified on the fly to match the context in which they are replayed. Beginning efforts in this direction may be found in [10].

8 Conclusions

We have described a system for off-line production and on-line playback of human figure motion for 3D virtual environments. The techniques employed are straightforward, and build upon several well known software systems and capabilities. As the number of possible states for a simulated human increases, the posture graphs will need to be replaced with a more procedural approach to changing posture. For applications built today on current workstations, the current technique is a balance between performance and realism.

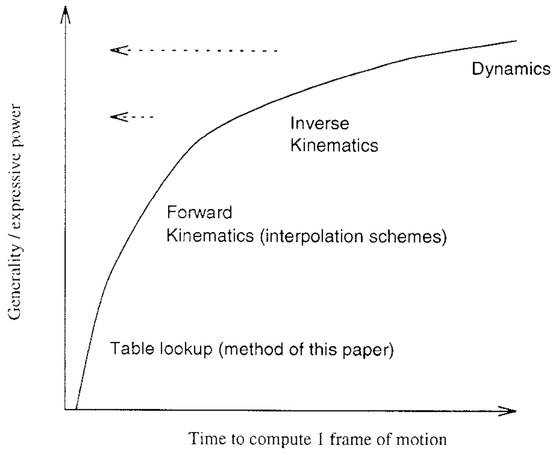

Figure 9: Trade-off between time and generality for motion generation techniques

Figure 9 shows a very coarse, and albeit intuitive, plot of the trade-offs between generality and computation time for several motion generation techniques. For realistic agent animation in virtual environments, the research community will be trying to push this curve to the left, making the more powerful techniques run faster. The curve has been drifting to the left in recent years mainly on the progress made in rendering hardware and overall workstation compute performance.

We chose humans for animating, as they are what we are interested in, but the techniques described in this paper could be applied to other complex articulated figures, whose states can be characterized by postures, and whose motions between postures can be organized into posture graphs.

Acknowledgments

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory (Aberdeen), Natick Laboratory, and NASA Ames Research Center; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; Naval Training Systems Center N61339-93-M-0843; Sandia Labs AG-6076; DMSO through the University of Iowa; NASA KSC NAG10-0122; MOCO, Inc.; NSF IRI91-17110, CISE CDA88-22719, and Instrumentation and Laboratory Improvement Program #USE-9152503.

References

[1] Norman I. Badler, Rama Bindiganavale, John Granieri, Susanna Wei, and Xinmin Zhao. Posture interpolation with collision avoidance. In *Proceedings of Computer Animation '94*, Geneva,

Switzerland, May 1994. IEEE Computer Society Press.

[2] Norman I. Badler, Cary B. Phillips, and Bonnie L. Webber. *Simulating Humans: Computer Graphics, Animation, and Control.* Oxford University Press, June 1993.

[3] Jay Banchero. Results to be published on system for dismounted infantry motion in a SIMNET image generator. Topographical Engineering Center, US Army.

[4] Edwin H. Blake. A metric for computing adaptive detail in animated scenes using object-oriented programming. In G. Marechal, editor, *Eurographics '87*, pages 295–307. North-Holland, August 1987.

[5] David R. Pratt et al. Insertion of an Articulated Human into a Networked Virtual Environment. In *Proceedings of the 1994 AI, Simulation and Planning in High Autonomy Systems Conference*, University of Florida, Gainesville, 7–9 December 1994.

[6] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 247–254, August 1993.

[7] Institute for Simulation and Training, Orlando, FL. *Standard for Distributed Interactive Simulation – Application Protocols (v 2.0, 4th draft, revised)*, 1993.

[8] Institute for Simulation and Training, Orlando, FL. *Enumeration and Bit-encoded Values for use with IEEE 1278.1 DIS - 1994*, ist-cr-93-46 edition, 1994.

[9] Hyeongseok Ko. *Kinematic and Dynamic Techniques for Analyzing, Predicting, and Animating Human Locomotion.* PhD thesis, University of Pennsylvania, 1994.

[10] Ken Perlin. Danse interactif. SIGGRAPH Video Review, Vol. 101 1994.

[11] Douglas A. Reece. Extending DIS for Individual Combatants. In *Proceedings of the 1994 AI, Simulation and Planning in High Autonomy Systems Conference*, University of Florida, Gainesville, 7–9 December 1994.

[12] John Rohlf and James Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 381–395, July 1994.

[13] Neil J. Rubenking. The DOOM Phenomenon. *PC Magazine*, 13(19):314–318, 1994.

[14] Greg Turk. Re-tiling polygonal surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[15] Frank Wysocki and David Fowlkes. Team Target Engagement Simulator Advanced Technology Demonstration. In *Proceedings of the Individual Combatant Modeling and Simulation Symposium*, pages 144–190, 15–17 February 1994. Held in Fort Benning, GA.

[16] Jeffrey Adam Young. Doom's Day Afternoon. *Computer Player*, pages 20–28, October 1994.

[17] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics, to appear*, 1995.