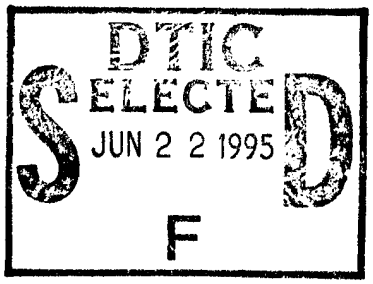
Center for Human Modeling and Simulation
Quarterly Progress Report
No. 54

Norman I. Badler
Director, HMS
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389
Fourth Quarter 1994

February 24, 1995



REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995	

4. TITLE AND SUBTITLE
CENTER FOR HUMAN MODELING AND SIMULATION QUARTERLY PROGRESS REPORT NO. 54

5. FUNDING NUMBERS
DAAL03-89-C-0031

6. AUTHOR(S)
Dr. Norman I. Badler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Pennsylvania
Computer & Information Science Department
Philadelphia, PA 19104-6389

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 26779.46-MA-AI

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This Quarterly Report includes descriptions of various projects underway at the Center for Human Modeling and Simulation during October through December.

14. SUBJECT TERMS
Human Modeling and Simulation

15. NUMBER OF PAGES
117

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

MEMORANDUM OF TRANSMITTAL

U.S. Army Research Office
ATTN: AMXRO-RT-IPL (Hall)
P.O. Box 12211
Research Triangle Park, NC 27709-2211

____ Reprint (15 copies) XX Technical Report (40 copies)

____ Manuscript (1 copy) ____ Final Report (40 copies)

____ Thesis (1 copy)

____ MS ____ PhD ____ Other_____

CONTRACT/GRANT NUMBER DAAL03-89-C-0031

TITLE: Center for Human Modeling and Simulation Quarterly Progress Report No. 54

is forwarded for your information.

SUBMITTED FOR PUBLICATION TO (applicable only if report is manuscript):

Sincerely,

Dr. Norman I. Badler
Professor, CIS and Director, HMS
University of Pennsylvania
Computer and Information Science Department
Philadelphia, PA 19104-6389

Contents

1 Introduction: Norman I. Badler

This Quarterly Report includes descriptions of various projects underway in the Center for Human Modeling and Simulation during October through December 1994.

These reports include:

- Release of the *Jack Motion Library*, and progress in motion authoring tools.
- Motion data acquisition to generate human animation.
- An update on SASS.
- The incorporation of strength analysis into motion planning.
- Development of fast inverse kinematics.
- Implementation of a grasping taxonomy for *Jack*
- Continued work in object specific reasoning.
- Improvements in the human body segment shapes, including a better hand model.
- Progress in adding dynamics to *Jack*.
- The *Zaroff* system and locomotion reasoning.
- The animation of fluid phenomena.
- Physics-based graphical modeling for respiratory mechanics.
- Updates to the *Jack* Motion Library.
- Continued work on space rendering.

There are also eight appendices:

- *Inverse Kinematics of the Human Arm*: Tolani
- *Adaptive Deformable Model Evolution Using Blending*: DeCarlo and Metaxas
- *Integrating Anatomy and Physiology for Behavior Modeling*: DeCarlo, Kaye, Metaxas, Clarke, Webber and Badler. Presented at the First International Symposium for Medical Robotics and Computer Assisted Surgery.
- *Volumetric Deformable Models with Parameter Functions: A New Approach to the 3D Motion Analysis of the LV from MRI-SPAMM*: Park, Metaxas, and Axel
- *Jack Reaching Planning With Strength Analysis and Collision Avoidance - User's Guide*: Xinmin Zhao
- *Behavioral Control for Real-Time Simulated Human Agents*: Granieri, Becket, Reich, Crabtree, and Badler; to appear in the 1995 Symposium on Interactive 3D Graphics.

- *Planning and Terrain Reasoning*: Moore, Geib, and Reich; to appear in the AAAI Spring Symposium on Integrated Planning Applications proceedings in 1995.
- *Production and Playback of Human Figure Motion for 3D Virtual Environments*: Granieri, Crabtree, and Badler; to be presented at the 1995 Virtual Reality Annual International Symposium (VRAIS '95).

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory and Natick Laboratory; ARPA AASERT DAAH04-94-G-0362; DMSO DAAH04-94-G-0402; ARPA DAMD17-94-J-4486; 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; NASA KSC NAG10-0122; MOCO, Inc.; National Library of Medicine N01LM-43551; DMSO through the University of Iowa; and NSF IRI91-17110, CISE CDA88-22719.

2 *Jack*: John Granieri

By the end of 1994, I finished the encapsulation of the DI (dismounted infantry) motion playback system (which we refer to informally as *JackML*, or *Jack Motion Library*, or simply *the motion library*) into a single object library, which can be linked and run within an IRIS Performer-based image-generator application (in this case it's currently used in TTES at NAWCTSD and in NPSNET-IV at Naval Post Graduate School.) (NPSNET is also distributed to various sites within the government.)

A paper entitled **Production and Playback of Human Figure Motion for 3D Virtual Environments**, to be presented at VRAIS '95 (See Appendix H), essentially describes the current state of the implementation, as well as some of the components of the library we're currently working on.

I plan on incorporating the fast inverse kinematics work being done by Deepak Tolani into *Jack* ML, as well as the work being done by Rama Bindiganavale.. This extends the functionality of *Jack* ML from a simple motion playback system to a more hybrid motion generation system.

I am currently building the motion level-of-detail (LOD) into *Jack* ML. This should improve performance of the visual system (so one can incorporate more soldiers in a simulation), as well as increase fidelity of ballistics (as they are computed via intersections, so the intersections can take place on the highest LOD human model).

The playback system relies on the notion of a *posture graph* to store and retrieve motion data. For the static posture changes we simply traverse the posture graph. For locomotion and crawling, we have a simple hard-coded state machine to map from a *state vector* (in the case of TTES, this is just the Entity State PDU lifeform fields) to motion generation. In the spirit of making an open, authorable motion generator, we wish to have this state machine also be authorable. To this end, we will investigate the use of PaT-Nets to author the mapping from the state vector to motion playback/generation. This gives us a very general purpose method for changing the motion control of the human figure. Since PaT-Nets are interpreted, it also allows the user to experiment. We will need to compile the PaT-Nets to a faster run-time version for execution within *Jack* ML.

Currently, the off-line motion authoring tools (for building posture graphs, as well as the state vector mapping functions) are built using a version of *Jack* and some C functions. The on-line motion tools are embedded in *Jack* ML, which in turn must be hosted in an IRIS Performer-based application. To unify these two separate environments, so one can build *and* run the motions (and motion generation techniques) in the same environment, I have begun the development of a new system which is in essence *Jack* running on top of IRIS Performer, with a Tk user interface. The user interface will allow us to build 2D interface components which will make it much easier to visualize and manipulate posture graphs and PaT-Nets. Some of this work is related to, and will be used by, other projects here.

The initial design for this multi-processing framework for behavioral simulation is described in the paper **Behavioral Control for Real-Time Simulated Human Agents** (See Appendix F) . We will be implementing the first cut of this system in the near term.

The key benefits for our research, from this unification, are (1) Since all current image-generation applications we are involved with are based on IRIS Performer, our code must be optimized for that API, (2) the multi-processing framework is needed to take advantage of the multiple processor machines which are coming to dominate the visual simulation field, (3) much of the work done already in *Jack* for controlling multiple figures and motion generation,- will then be available for integration into *Jack* ML. This system will be used to prototype and build the control functions for the real-time simulated agents which will be used in distributed simulations for our sponsors under the DMSO project.

This new system will most likely become the standard *Jack* system environment in the future.

3 Motion Data Acquisition: Mike Hollick

We have begun work on a system that will be used to gather motion data for generating human animation, as well as real-time interaction. The first step has been to purchase and integrate 4 additional Flock of Birds sensors. By adding these to our current Flock we can begin to experiment with sensor placement configurations to increase the accuracy of the recorded posture data. In order to support the additional sensors, the current driver has been modified and tested off-site to confirm that the sensor communication will work correctly through an intermediate hardware interface such as a terminal server. This intermediate interface is needed to maintain a direct serial connection with each sensor when the number of sensors exceeds the available serial ports on the host workstation.

The next step is to determine sensor placements that produce the most accurate representations of the actual posture, while minimizing the degree of encumbrance. This will also involve work on the *Jack* side of the system, where we will need to test different constraint types and heuristics to use the data most efficiently. It is probable that several sensor configurations will be ultimately selected, with each configuration being the best for a certain class of motion. For example, for motions that only involve the upper body, it would make little sense to have sensors on the knees and ankles.

4 SASS: Francisco Azuola

During the last quarter, I finished the implementation of SASS v.2.5 to be released with *Jack* 5.9. The SASS user's manual will be provided with the new *Jack* 5.9 manual.

Omission

In Quarterly Progress Report No.51 and in the articles, "Infrastructure for Human Modelling in VR" and "Building Anthropometry-Based Virtual Human Models", we omitted to mention the contribution of Dr. Ann Aldridge (from NASA) to the figure scaling project, namely, the original version of the "anthropometric data extraction" tool.

5 Motion Planning with Strength Analysis: Xinmin Zhao

For the past several months, I have been working on incorporating strength analysis into the motion planning process. The objective is that the planned motion should be not only collision free, but also strength feasible for an agent with limited strength. The motion planning algorithm has been modified to include strength analysis. Our experiments with the modified algorithm so far show that it works well. For more information, please see the user's guide of *Jack* reach planning (Appendix E).

6 Inverse Kinematics of the Human Arm: Deepak Tolani

My current task in the DMSO project is the generation of inverse kinematics for the human arm. In particular, I've focused on two separate problems: (1) generating inverse kinematics solutions in real time using a simplified model of the human arm, and (2) calculating accurate inverse kinematics using *Jack*'s model of the shoulder-arm complex. Appendix A summarizes my current status in these two areas.

7 Grasping Implementation: Brett J. Douville

During the past three months I have been working on the implementation of a grasping taxonomy for *Jack*. Once this work has been completed *Jack* will be able to reach for an object, grasp it in an appropriate grasp, and then manipulate the object.

The grasping behaviors are currently programmed in Lisp using PaT-Nets. Individual instantiations of finger-controlling and thumb-controlling PaT-Nets generate realistic behavior by directly manipulating joint angles. Collisions between the segments of the finger (or thumb) and other fingers or the object to be grasped determine transitions to different states of the PaT-Net.

The motivation behind using collision detection to drive the grasping behavior is based in human grasping tasks: when humans grasp an object they are searching largely for *tactile* information: by driving grasping using collision detection, *Jack* is simulating tactile sensations. I expect to distinguish between several different types of grasping collisions in the upcoming months, which should lead to even better grasping.

Currently, grasping is being integrating with Xinmin Zhao's motion planning to achieve realistic reaching and grasping.

8 Object Specific Reasoning: Libby Levison

In the last quarter I finished writing my dissertation proposal [1] and began implementing the system.

Currently there is a complete vertical integration for a subset of the task-action commands: the actions LOOK, GRASP, RELEASE and REACH are fully implemented. The OSR can output *Jack* action directives of various agent performing these actions on various objects. The OSR decomposes the actions, and determines missing information needed by the *Jack* system (eg, which hand to use). It then checks that the agent has adequate resources to perform the actions, and outputs the directives.

In addition to the algorithm which reasons about each task-action and converts it to a set of action directives, the vertical integration has required the construction of two additional components. First is a knowledge base of properties of the agents and objects which the system uses in determining the missing information. Second, I have built the task-action library for the task-actions listed above; the library contains the underspecified definitions of these commands.

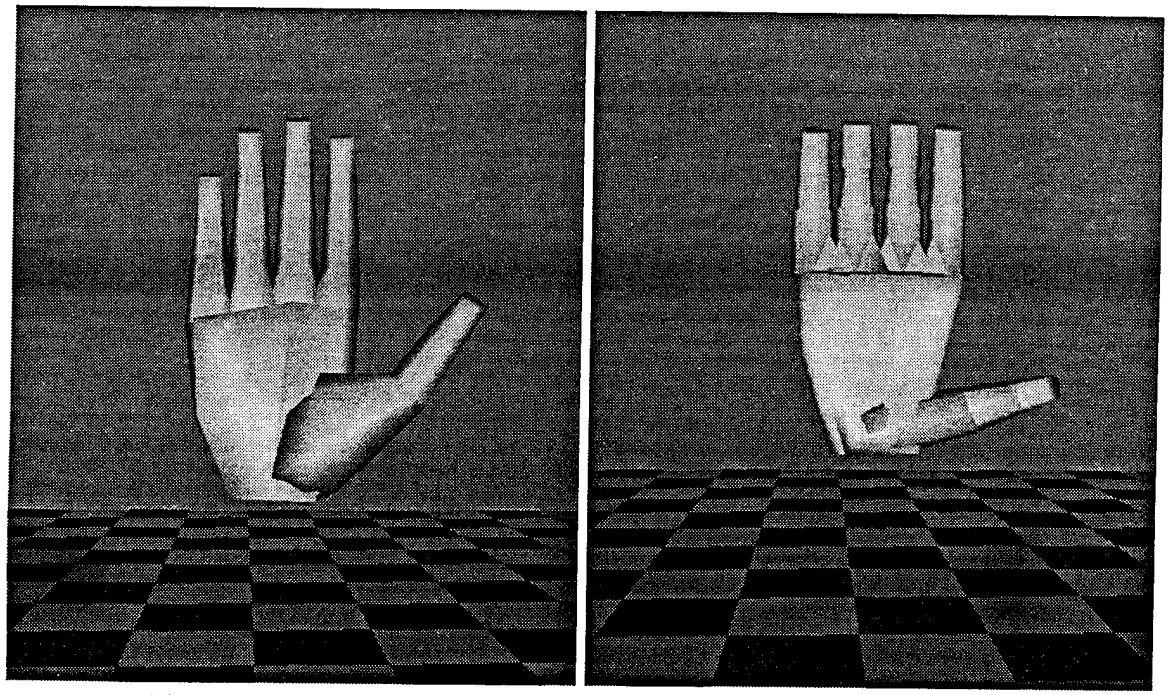
In the next quarter, I plan to continue work on the implementation. In addition to encoding new actions (eg MOVE-OBJECT and LOCOMOTE, the OSR system must be integrated into *Jack*. This includes 1) converting the knowledge base to interface with the graphics knowledge base, which will allow the OSR to acquire as much knowledge as possible from the graphics knowledge base, and 2) generating action directives which are *Jack* behaviors, and performing them in the *Jack* system. By the end of Q1.95 I plan to have a stable system implementation.

9 Improvement of Human Model: Bond-Jay Ting

When human 5.8 was created, the major work was on the head and neck. Although there were some minor improvements, the hand model didn't get much attention. Therefore, in this version of human model improvement, the focus is on creating a better hand model.

To minimize the impact on *Jack* the redesign is based on the old segment structure. The total number of segments remains 16. Although there is a slight change in the joint position of the palm, the rest of the joints were kept at the original position. The major change is the geometry for each segment:

[1] "Connecting Planning and Acting: Towards an Architecture for Object Specific Reasoning" (1995), Libby Levison, University of Pennsylvania.



(a) The new hand model (b) Hand model for *Jack* 5.8

Figure 1: Hand model

Palm: Palm is the most complicated segment in the hand model. After studying the muscles and the joint locations of the palm, the outside part of the hand is made to be a more rounded shape. And instead of the straight line distribution in the previous version, the new joint locations for fingers have been moved to a more rounded fashion.

Fingers: Basically, the shapes of the fingers remained the cylindrical cone shape. The limited changes are located in the proximal parts (the first segments) and the distal parts (the last segments). For the proximal parts, the fins have been enlarged and moved toward the front part of the hand. For the distal parts, the shapes have been redesigned to meet the shape of finger tips.

Thumb: One of the biggest changes of the new hand model is the thumb. The first segment of thumb has been redesigned with a larger base to simulate the part of the segment which is embedded in the palm. Also, similar to the distal part of the fingers, the tip of the thumb has been redesigned to simulate the shape of thumb.

Fig. 1 shows both the new and old hand model. The new hand model also uses polygons more efficiently. Compared to 468 polygons in the previous version, the new hand model has only 391 polygons.

To create a physics based hand motion, we need to insert the mass properties of the hand into the model. Unfortunately, the current mass data collected for the hand model are either "total hand mass" or "total mass for fingers and the mass for palm". Neither case is sufficient enough for our simulation. Therefore, a distribution function is needed to distribute the mass into different

segments. Since there is no data showing the density of each segment, we make the uniform density distribution assumption. The mass is distributed according to the volume of each segment.

Beside the hand modeling, I have also been working on the uneven scaling problem. In *Jack*, the scaling problem is handled by a uniform scaling scheme. Each node in the segment is scaled according to a three dimensional vector in which each component represents the scaling factor in the corresponding axis. When creating a new human figure from one prototype to match a set of measured data, the scaling factor in two ends of a segment don't necessarily have to be the same. That is, the uniform scaling scheme is not sufficient.

To solve this problem, an uneven scaling scheme is introduced. Uneven scaling scales along a predefined axis (one of the local coordinate axes or global coordinate axes). The scaling vector of any node in the segment is linearly interpolated according to two reference positions (nodes or sites) and two corresponding scaling vectors.

10 Recursive Forward Dynamics Algorithm: Evangelos Kokkevis

After testing the recursive forward dynamics algorithm in a simple simulation environment, the code is now incorporated into *Jack*. An effort has been made to create a general system that can automatically generate dynamically correct motion for any articulated structure defined as a standard *Jack* figure. The only extra parameter that user needs to supply is the mass of each segment. The moment of inertia and the center of mass are automatically computed from the segment's geometry. The resulting system is both intuitive to use and runs at interactive speeds, even for fairly complex structures. To detect collisions between objects, the fast collision detection algorithm already implemented in *Jack* has been employed. The user can specify the coefficients of friction and restitution for the objects and hence, simulate different material properties, enhancing the realism of the animation.

With a general dynamical simulation environment at hand, the next step was to investigate ways to give the user control over the animation. The goal would be to have the dynamics assisting the user in creating a realistic animation without overrestricting him on the motions he can generate. For a specific motion, the user should be able to prescribe the trajectory of the important (to the motion) degrees of freedom and leave the state of the rest of the figure to be handled naturally by the dynamic simulator. A dynamical controller should then be employed to generate the internal forces and torques needed to be exerted on the joints to have them follow the specified trajectories. Model Reference Adaptive Control proved to be a good choice for the controllers. Adaptive controllers have the advantage, as the name suggests, to progressively "learn" the dynamic properties of the system they control. They are also simple to implement and run efficiently since no knowledge of the often complicated underlying system dynamics needs to be hardwired into them. At this stage, kinematic joint trajectories provided by the user can be replicated dynamically using this type of feedback control.

Adaptive control properties need to be investigated further to give a more robust general animation system. Once this is done, the next step will be to apply the above system in the more specific

application of dynamically controlling human motion.

11 Locomotion Reasoning: Barry D. Reich

This quarter I worked on the *Zaroff* system, an animated simulation of a game of Hide and Seek, with Chris Geib, Mike Moore, and Tripp Becket. We use *Jack* to create an architecture where AI planning can be combined with sensor-based, reactive navigation. In the PaT-Net-controlled simulations, planning is used to generate intentions. The intentions are achieved through the use of PaT-Nets which configure a set of simulated sensors to execute the desired actions.

We wrote a paper describing the Zaroff system which will appear in the *AAAI Spring Symposium on Integrated Planning Applications* proceedings in 1995 (also University of Pennsylvania CIS Department Technical Report MS-CIS-93-56/LINC LAB 280). It is included in this report (See Appendix G). We also worked on papers for *The 5th Conference on Computer Generated Forces and Behavioral Representation* and *The International Joint Conference on Artificial Intelligence*. The former has since been accepted.

This quarter I also worked on a project and paper for the 1995 *Symposium on Interactive 3D Graphics* with John Granieri, Tripp Becket, and Jonathan Crabtree (See Appendix F). We are developing a system for interactive behavioral programming in real-time.

12 Realistic Animation of Liquids: Nick Foster

Techniques developed for modeling, and rendering fluids were combined as a single development system for animating liquid effects [2] . The system provides a user interface for interactively modeling a complex environment including liquid, obstacles and floating objects. The behavior of the liquid and objects in the scene can be calculated accurately using a physics-based model, and the results rendered realistically using the RenderMan interface [3]. The system was used for a number of sample applications. These included an animation of ocean waves, and a simulation of internal bleeding from a penetrating knife wound to the lung.

13 Modeling Respiratory Mechanics: Jonathan Kaye

The models of respiratory mechanics I developed earlier showed the qualitative behavior of the system for normal, quiet breathing. The motivation of developing these was to demonstrate basic, qualitative relationships during breathing. While these simulations were sufficient to show basic behavior, thus having potential for explaining why pressures and volumes change, we needed more precision to drive the physics-based graphical modeling (for visualization).

[2] Realistic Animation of Liquids, N. Foster and D. Metaxas. Submitted to SIGGRAPH 1995.
[3] The RenderMan Companion, S. Upstill; Addison Wesley, New York, (1990)

During the current reporting period, I presented a paper on our work at the *First International Symposium for Medical Robotics and Computer Assisted Surgery* (See Appendix C).

Needing more detail than the qualitative models provided, I reworked my models and derived ordinary differential equations (quantitative) for different pathological conditions on multiple compartment lung models (e.g., trapped air in pleural space, simple pneumothorax, open sucking chest wound, and tension pneumothorax). In the process, I learned more about respiratory mechanics to validate the approach. I am planning to incorporate these new models with cardiovascular modeling, to show how the physiological systems are dependent in some situations because of the physical space they share.

14 *Jack* Motion Library: Jonathan Crabtree

The greater part of the work done during this quarter relates to the DMSO project, either in the form of continued support for the NAWCTSD TTES system or in the form of preliminary investigations into those areas spanned by the first-year report deliverables.

Continued support for the TTES application has been comprised primarily of updates to the *Jack* Motion Library. In particular, all the application code was modified for compatibility with SGI Performer 1.2, the latest version of the rendering toolkit on which both TTES and DMSO will ultimately depend for their real-time display requirements. Updates to the *Jack* Motion Library include bug fixes and optimizations for greater efficiency, most notably the addition of an option to fully precompute all the joint transformation matrices involved in any given figure motion. This will almost certainly be an essential component in maintaining the real-time constraints inherent in the second-year DMSO deliverables.

As a first step toward determining what changes will be necessary in human agent PDUs (Protocol Data Units) for this project, a more general mechanism for making posture changes was explored and implemented; while the posture graph establishes a regimen on the organization and storage of motion data, it does not directly address issues of how information about **when** posture changes take place should be stored. For instance, in cyclic traversals of a posture graph, such as arise in animating walking or running, it is desirable to associate conditions with edges of the posture graph, indicating that an agent must continue to make posture changes (for instance, a walking agent in the TTES system) as long as that agent's velocity is nonzero. The possible sets of edge conditions will correspond closely with the information encodeable in an expanded human agent PDU; at the very minimum, each action that can be animated must be representable in the distributed protocol, and conversely each distinct message under the protocol should map to some transition or set of transitions and conditions in the posture graph.

Future work in this area will include an analysis of the movement set (once it has been fully realized) to extract a behavioral hierarchy. Knowledge of this hierarchy will guide further extensions (or development of additional structures) to the posture graph abstraction in order to represent behavioral state information. We have also been exploring, at the implementation level, the possibility of segmenting motion by body region. This would allow for multiple concurrent behaviors; for instance, an agent might be walking while carrying an object or while aiming a weapon or operating a tool.

In realted work, a database converter was developed to output the contents of a SGI Performer visual database into a *Jack*-readable format (a collection of figure and psurf files). The conversion process preserves articulation information and also texture and color specifications. In conjunction with the wide range of database loaders available for Performer 1.2, this provides a path, through the intermediate Performer format, to convert files of several different types (.flt, .sgo, .pto, etc.) into the *Jack* peabody language.

A new display mode was added to the standard version of *Jack*. It aids in visualizing the "skeletons" of figures by drawing a small sphere at each joint center, and rectangular "sticks" for the segments between joints. This mode could be expanded to provide a simpler and more intuitive interface for joint manipulation and figure positioning, by making the precise locations of all joint centers visible simultaneously. With the standard figure display disabled, visual clutter is also reduced.

15 Efficient Rendering: Jeff Nimeroff

During the last quarter Julie Dorsey, Eero Simoncelli, Norman Badler and myself completed work on the final draft of our paper on rendering spaces [4] that was accepted in Presence, the Journal of Virtual Reality and Teleoperators. The paper abstracted specific rendering scenerios into subspaces of a general rendering space and was an attempt at applying algebraic abstraction techniques to computer graphics. Julie Dorsey and I also completed a SIGGRAPH submission on photorealistic rendering based on research that is to be continued throughout the next two quarters.

[4] "Rendering Spaces for Architectural Environments", J. Nimeroff and J. Dorsey and E. Simoncelli and N. Badler Accepted for Publication in *Presence*, the Journal of Virtual Reality and Teleoperators, November 1994.

A Inverse Kinematics of the Human Arm: Deepak Tolani

Inverse Kinematics of the Human Arm

Deepak Tolani

February 21, 1995

1 Introduction

My current task in the DMSO project is the generation of inverse kinematics for the human arm. In particular, I've focused on two separate problems: (1) generating inverse kinematics solutions in real time using a simplified model of the human arm, and (2) calculating accurate inverse kinematics using Jack's model of the shoulder-arm complex. This report summarizes my current status in these two areas and outlines

2 Fast Inverse Kinematics

The best existing numerical algorithms for inverse kinematics are only marginally adequate for real time applications. Additionally, most numerical methods don't yield all solutions and they often fail near a singularity of the manipulator. For these reasons, it is desirable to obtain an analytical solution. One of the conditions which guarantees a closed-form solution in a six degree of freedom manipulator is the presence of three intersecting joint axes. Thus, if we utilize a simplified model of the human arm where the shoulder is modeled as a spherical joint, we should be able to derive an analytical solution.

Figure 1 illustrates the Peabody representation of a simplified version of Jack's left arm. In the simplified model, the shoulder-clavicle complex is reduced to a single spherical joint with three degrees of freedom. The joint transformations from the shoulder to the wrist frame are given by

$$R_z(\phi_1)R_x(\phi_2)R_z(\phi_3)T_1R_y(\phi_4)T_2R_y\,(\phi_5)\;R_x(\phi_6)R_z(\phi_7)$$

$$T_1 = \begin{bmatrix} & -a_3 \\ \mathbf{I} & 0 \\ & d_3 \\ 0 & 1 \end{bmatrix} \qquad T_2 = \begin{bmatrix} & 0 \\ \mathbf{I} & 0 \\ & a_4 \\ 0 & 1 \end{bmatrix}$$

$$a_3 = .49, d_3 = 32.8, a_4 = 25.93$$

where T_1 and T_2 are constant matrices that relate the positions of the shoulder,elbow, and wrist joints. Note that the equation above assumes column notation for vectors.

The inverse kinematics problem may be stated as follows. Given a desired position and orientation of the wrist frame relative to the shoulder frame $\mathbf{A}_{wrist}$ find a suitable set of angles $\theta_1, ..., \theta_7$ such that the following equation is satisfied:

$$\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6\mathbf{A}_7 = \mathbf{A}_{wrist} \tag{1}$$

Where

$$A_1 = \begin{bmatrix} \cos(\theta 1) & -\sin(\theta 1) & 0 & 0 \\ \sin(\theta 1) & \cos(\theta 1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta 2) & -\sin(\theta 2) & 0 \\ 0 & \sin(\theta 2) & \cos(\theta 2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} \cos(\theta 3) & -\sin(\theta 3) & 0 & -1.\cos(\theta 3)\, a3 \\ \sin(\theta 3) & \cos(\theta 3) & 0 & -1.\sin(\theta 3)\, a3 \\ 0 & 0 & 1 & d3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} \cos(\theta 4) & 0 & \sin(\theta 4) & \sin(\theta 4)\, a4 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta 4) & 0 & \cos(\theta 4) & \cos(\theta 4)\, a4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} \cos(\theta 5) & 0 & \sin(\theta 5) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta 5) & 0 & \cos(\theta 5) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_6 = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta 6) & -\sin(\theta 6) & 0 \\ 0 & \sin(\theta 6) & \cos(\theta 6) & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_7 = \begin{bmatrix} \cos(\theta 7) & -\sin(\theta 7) & 0 & 0 \\ \sin(\theta 7) & \cos(\theta 7) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$a3 = .49 \quad d_3 = 32.8 \quad a_4 = 25.93$$

However, $\mathbf{A}_{wrist}$ specifies only six independent equations but we have seven unknowns. Thus, in general, there are an infinite number of joint angles satisfying equation 1. The simplest solution is to forfeit one degree of freedom of the system. Since many positioning tasks do not simultaneously utilize all three degrees of freedom of the wrist, it is often expedient to fix one of the wrist joints to a rest angle.

2.1 Case 1: Joint angle 7 is constant

Without loss of generality, assume that the last wrist joint is constrained so that A_7 is a constant matrix. We can then write the inverse kinematics problem as

$$\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6 = \mathbf{A}_{wrist}\mathbf{A}_7^{-1} \tag{2}$$

where the right hand side is given and the unknowns are the joint angles $\theta_1, ..., \theta_6$.

Denoting

$$\mathbf{A}_{wrist} = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

, the vector $\mathbf{p} = [g_{14}, g_{24}, g_{34}]^T$ is the origin of the wrist frame measured in the shoulder coordinate system. The magnitude of $\mathbf{p}$ depends only upon the lengths of the upper and lower arms and θ_4. As shown in figure 3, the relationship between θ_4 and $||\mathbf{p}||$ can be derived from the law of cosines as

$$\begin{aligned} \theta_4 &= \pi - \psi - \alpha \\ \alpha &= \arctan(\frac{a_3}{d_3}) \\ \psi &= \arccos(\frac{l^2 + a_4^2 - ||\mathbf{p}||^2}{2l\, a_4}) \\ l &= \sqrt{a_3^2 + d_3^2} \end{aligned} \tag{3}$$

Since $\mathbf{p}$ is given, θ_4 can be calculated directly from equation 3. Although are two distinct solutions for θ_4, only one answer is physically realizable because of joint limits.

To calculate the remaining joint angles, we note that the position of the shoulder joint expressed in coordinate system six is just a function of the last three joints $\theta_4, \theta_5, \theta_6$. More precisely, we can write

$$\begin{aligned} {}^6\mathbf{p} &= \mathbf{A}_6^{-1}\mathbf{A}_5^{-1}\mathbf{A}_4^{-1}\mathbf{A}_3^{-1}\mathbf{A}_2^{-1}\mathbf{A}_1^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ &= \mathbf{A}_6^{-1}\mathbf{A}_5^{-1} \begin{bmatrix} \cos(\theta_4)a_3 + \sin(\theta_4)d_3 \\ 0 \\ \sin(\theta_4)a_3 - \cos(\theta_4)d_3 - a_4 \end{bmatrix} \end{aligned} \tag{4}$$

We also note that we can compute ${}^6\mathbf{p}$ as

$${}^6\mathbf{p} = \left(\mathbf{A}_7\mathbf{A}_{wrist}^{-1}\right) \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5}$$

$$\begin{pmatrix} {}^6p_x \\ {}^6p_y \\ {}^6p_z \end{pmatrix} = \begin{bmatrix} -\cos(\theta_7)t1 + \sin(\theta_7)t2 \\ -\sin(\theta_7)t1 - \cos(\theta_7)t2 \\ -(g13g14 + g23g24 + g33g34) \end{bmatrix} \tag{6}$$

where

$$\begin{aligned} t_1 &= g11g14 + g21g24 + g31g34 \\ t_2 &= g12g14 + g22g24 + g32g34 \end{aligned}$$

Equating equations 4 and 5 gives

$$\begin{bmatrix} \cos(\theta_4)a_3 + \sin(\theta_4)d_3 \\ 0 \\ \sin(\theta_4)a_3 - \cos(\theta_4)d_3 - a_4 \end{bmatrix} = \mathbf{A}_5\mathbf{A}_6 \begin{pmatrix} {}^6p_x \\ {}^6p_y \\ {}^6p_z \\ 1 \end{pmatrix} \tag{7}$$

where the only unknowns are θ_5 and θ_6. Denoting the left hand side of equation 7 by $[f_1, 0, f_3]^T$ and expanding the right hand side yields three scalar equations

$$\begin{aligned} p_x\cos(\theta_5) + p_y\sin(\theta_5)\sin(\theta_6) + p_z\cos(\theta_6)\sin(\theta_5) &= f1 \\ p_y\cos(\theta_6) - p_z\sin(\theta_6) &= 0 \\ -p_x\sin(\theta_5) + p_y\cos(\theta_5)\sin(\theta_6) + p_z\cos(\theta_5)\cos(\theta_6) &= f3 \end{aligned} \tag{8}$$

The second equation is of the form

$$a\cos(\theta_6) + b\sin(\theta_6) = c$$

and has two solutions given by

$$\theta_6 = a\tan 2(b, a) \pm a\tan 2(\sqrt{a^2 + b^2 - c^2}, c)$$

Substituting a value for θ_6 into each of the first two equations of system 8 yields a set of equations of the form

$$\begin{aligned} a\cos(\theta_5) - b\sin(\theta_5) &= c \\ a\sin(\theta_5) + b\cos(\theta_5) &= d \end{aligned}$$

which has the solution

$$\theta_5 = a\tan 2(ad - bc, ac + bd)$$

where atan2 is the two argument arctangent function. Finally, the values of $\theta_1, \theta_2, \theta_3$can be determined by extracting the Euler angles from the rotational components of $\left(\mathbf{A}_{wrist}\mathbf{A}_7^{-1}\right)\mathbf{A}_6^{-1}\mathbf{A}_5^{-1}\mathbf{A}_4^{-1}$. If we rearrange equation 1 as

$$\mathbf{A}_1\mathbf{A}_2\mathbf{A}_3 = \left(\mathbf{A}_{wrist}\mathbf{A}_7^{-1}\right)\mathbf{A}_6^{-1}\mathbf{A}_5^{-1}\mathbf{A}_4^{-1} \tag{9}$$

we note that the right hand side contains quantities that have been determined. Denoting the 3×3 rotational component of the right hand side of the matrix equation by $\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ and using c_i and s_i to represent $\cos(\theta_i)$ and $\sin(\theta_i)$ yields the equations

$$\begin{bmatrix} c_1c_3 - s_1c_2s_3 & -c_1s_3 - s_1c_2c_3 & s_1s_2 \\ s_1c_3 + c_1c_2s_3 & -s_1s_3 + c_1c_2c_3 & -c_1s_2 \\ s_2s_3 & s_2c_3 & c_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

We can obtain two possible values for θ_2 from the (3,3) component of the matrix equation above

$$\theta_2 = \pm \arccos(r_{33})$$

We can then solve for θ_3 and $\theta_1$by taking the appropriate arc tangents

$$\begin{aligned} \theta_3 &= \arctan 2\left(\frac{r_{31}}{\sin(\theta_2)}, \frac{r_{32}}{\sin(\theta_2)}\right) \\ \theta_1 &= \arctan 2\left(\frac{r_{13}}{\sin(\theta_2)}, \frac{-r_{23}}{\sin(\theta_2)}\right) \end{aligned}$$

where arctan2 is the two-argument version of the arc tangent function. If $\theta_2 = 0$ or 180 the equations for θ_3 and θ_1 degenerate. In this case, we can only compute their sum or difference. One possible workaround is to add a small ϵ to θ_2. Another fix is to arbitrarily set $\theta_3 = 0$ and to compute θ_1 as

$$\theta_1 = \arctan 2(r_{21}, -r_{11})$$

where arctan2 is the two argument arctangent function.

2.2 Case 2: Joint angle 5 is constant

In the previous section, we assumed that the last wrist joint was fixed. We now consider the case where the first wrist angle is constant, and joints 6 and 7 are allowed to move. We first compute θ_4 using the same technique as in the previous section. Let $\mathbf{p}$ represent the vector from the origin of the wrist frame to the origin of the shoulder frame. The coordinates of $\mathbf{p}$ measured in frame 5 can be calculated by the following two equations

$$
\begin{aligned}
{}^5\mathbf{p} &= \mathbf{A}_5^{-1}\mathbf{A}_4^{-1}\mathbf{A}_3^{-1}\mathbf{A}_2^{-1}\mathbf{A}_1^{-1}\begin{pmatrix}0\\0\\0\\1\end{pmatrix} \\
\begin{pmatrix}{}^5p_x\\{}^5p_y\\{}^5p_z\\1\end{pmatrix} &= \begin{bmatrix}\cos(\theta_5)(\sin(\theta_4)d_3+\cos(\theta_4)a_3)+\sin(\theta_5)(-\sin(\theta_4)a_3+\cos(\theta_4)d_3+a_4)\\0\\\sin(\theta_5)(\sin(\theta_4)d_3+\cos(\theta_4)a_3)+\cos(\theta_5)(-a4+\sin(\theta_4)a_3-\cos(\theta_4)d_3)\end{bmatrix}
\end{aligned}
$$

and

$$
{}^5\mathbf{p} = \mathbf{A}_6\mathbf{A}_7\mathbf{A}_{wrist}^{-1}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}
$$

equating these two equations gives

$$
\begin{aligned}
\sin(\theta_7)f_1+\cos(\theta_7)f_2 &= p_x \qquad (10)\\
f_1\sin(\theta_6)\cos(\theta_7)-f_2\sin(\theta_6)\sin(\theta_7)-f_3\cos(\theta_6) &= 0\\
f_1\cos(\theta_6)\cos(\theta_7)-f_2\cos(\theta_6)\sin(\theta_7)+f_3\sin(\theta_6) &= p_z
\end{aligned}
$$

where

$$
\begin{aligned}
f_1 &= -(g_{11}g_{14}+g_{21}g_{24}+g_{31}g_{34})\\
f_2 &= -(g_{12}g_{14}+g_{22}g_{24}+g_{32}g_{34})\\
f_3 &= -(g_{13}g_{14}+g_{23}g_{24}+g_{33}g_{34})
\end{aligned}
$$

The equation above is very similar to equation 8 and can be solved using a similar approach. Finally, once θ_6 and θ_7 have been computed, θ_1, θ_2,and θ_3 can be calculated by using the Euler angle extraction technique discussed in the previous section.

2.3 Case 3: Joint angle 6 is constant

Finally, consider the case where the middle wrist joint is fixed. Let ${}^7\mathbf{p}$ denote the coordinates of the shoulder joint measured in the final wrist frame. We have two alternative ways to compute ${}^7\mathbf{p}$:

$$ {}^7\mathbf{p} = \begin{pmatrix} {}^7p_x \\ {}^7p_y \\ {}^7p_z \\ 1 \end{pmatrix} = \mathbf{A}_{wrist}^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} $$

and

$$ {}^7\mathbf{p} = \mathbf{A}_7^{-1}\mathbf{A}_6^{-1}\mathbf{A}_5^{-1} \begin{bmatrix} \cos(\theta_4)a_3 + \sin(\theta_4)d_3 \\ 0 \\ \sin(\theta_4)a_3 - \cos(\theta_4)d_3 - a_4 \end{bmatrix} $$

Equating the two equations above and moving $\mathbf{A}_6$ and $\mathbf{A}_7$ to the right hand side yields the equations

$$ \begin{aligned} \cos(\theta_5)h_1 - \sin(\theta_5)h_3 &= \cos(\theta_7)f_1 - \sin(\theta_7)f_2 \qquad (11) \\ 0 &= \cos(\theta_7)(\cos(\theta_6)f_2) + \sin(\theta_7)(\cos(\theta_6)f_1) - \sin(\theta_6)f_3 \\ \sin(\theta_5)h_1 + \cos(\theta_5)h_3 &= \cos(\theta_7)(\sin(\theta_6)f_2) + \sin(\theta_7)(\sin(\theta_6)f_1) + \cos(\theta_6)f_3 \end{aligned} $$

where

$$ \begin{aligned} h_1 &= \cos(\theta_4)a_3 + \sin(\theta_4)d_3 \\ h_3 &= \sin(\theta_4)a_3 - \cos(\theta_4)d_3 - a_4 \end{aligned} $$

and

$$
\begin{aligned}
f_1 &= -(g_{11}g_{14} + g_{21}g_{24} + g_{31}g_{34}) \\
f_2 &= -(g_{12}g_{14} + g_{22}g_{24} + g_{32}g_{34}) \\
f_3 &= -(g_{13}g_{14} + g_{23}g_{24} + g_{33}g_{34})
\end{aligned}
$$

Since θ_6 is constant, the only unknowns are θ_5 and θ_7. The second equation of system 11 is of the form

$$a\cos(\theta_7) + b\sin(\theta_7) = c$$

and has two solutions given by

$$\theta_7 = a\tan 2(b, a) \pm a\tan 2(\sqrt{a^2 + b^2 - c^2}, c)$$

For a given value of θ_7, we can find the corresponding values of θ_5 by solving equations 1 and 3 of system 11 which are of the form

$$
\begin{aligned}
a\cos(\theta_5) - b\sin(\theta_5) &= c \\
a\sin(\theta_5) + b\cos(\theta_5) &= d
\end{aligned}
$$

with the solution

$$\theta_5 = a\tan 2(ad - bc, ac + bd)$$

3 Accurate Inverse Kinematics

The inverse kinematics formulas above assume that the shoulder is a simple three degree of freedom spherical joint. However, the actual shoulder-clavicle joint complex used in Jack is more complicated. In Jack, the shoulder complex consists of five interdependent joints, two of which are located at the clavicle and three of which are located at the shoulder. These five joints are controlled with three degrees of freedom: elevation, abduction, and twist (ϕ, θ, τ) corresponding approximately to a spherical motion. The coordinate transformation from the left elbow to the clavicle joint is given by the equation

$$C(\theta, \phi)T_1S(\phi, \theta, \tau)T_2 \qquad (12)$$

where $C(\theta,\phi)$ and $S(\phi,\theta,\tau)$ are transformations given by the clavicle and shoulder joints

$$\begin{aligned} C(\theta,\phi) &= R_y(\theta_1)R_x(\phi_1) \\ S(\phi,\theta,\tau) &= R_x(-\phi_1)R_z(\theta_2)R_x(\phi)R_z\left(\tau+\left(\frac{\phi}{90}-1\right)\theta_2\right) \end{aligned}$$

and

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_2 = \begin{bmatrix} 1 & 0 & 0 & -.49 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 32.8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} \phi_1 &= \cos(\theta)\beta_1 + (1-\cos(\theta))\beta_2 - 90 \\ \theta_1 &= \frac{\theta}{5} \quad \theta_2 = \frac{4\theta}{5} \end{aligned}$$

$$\beta_1 = \begin{cases} .251\phi + 97.076 & 0 \le \phi \le 131.4 \\ -.035\phi + 128.7 & otherwise \end{cases}$$

$$\beta_2 = \begin{cases} .21066\phi + 92.348 & 0 \le \phi \le 130.0 \\ 120 & otherwise \end{cases}$$

Note that these equations are **not** the same as the ones in the book "Simulating Humans". However, I believe they accurately reflect the transformations used in the current implementation of Jack. I will now discuss two possible methods to implement the inverse kinematics for the Jack shoulder-arm complex. The net transformation from the wrist to the clavicle joint is given by the rather formidable equation

$$\begin{aligned} & R_y(\theta_1)R_x(\phi_1)T_1R_x(-\phi_1)R_z(\theta_2)R_x(\phi)R_z(\tau')T_2R_y(\delta)R_y(\omega_1)R_x(\omega_2)R_z(\omega_3) \\ = \; & A_{wrist} \end{aligned}$$

where $\tau' = \tau + \left(\frac{\phi}{90}-1\right)\theta_2, \delta$ is the elbow angle, and $\omega_1, \omega_2, \omega_3$ are the wrist angles one of which is assumed to be constant. The inverse kinematics problem is to find a suitable set of joint angles $\theta, \phi, \tau, \delta, \omega_1, \omega_2, \omega_3$ that interpolates a desired A_{wrist}.

3.1 Method 1 : A Hybrid Analytic and Numerical method.

Equation 12 can be approximated very crudely by a coordinate transformation from the elbow joint to the shoulder joint given by :

$$R_z(\theta)R_x(\phi)R_z\left(\tau + \left(\frac{\phi}{90} - 1\right)\theta_2\right)T_2 \tag{13}$$

Since we already know how to analytically compute the inverse kinematics of a spherical mechanism, we can compute an approximate solution to equation 12 by solving for θ, ϕ, τ in equation 13. We can then input these angles as a "first guess" to a numerical procedure, such as Jack's own inverse kinematics routines, to obtain an accurate solution. There are two primary advantages of this hybrid approach over a purely numerical method. Most numerical methods can only generate a single solution. Moreover, not all inverse kinematic numerical methods are guaranteed to converge. Using an approximate analytical solution allows multiple solutions to be explored. Additionally, since the numerical procedure is invoked near a solution, convergence is both more rapid and probable.

However, there is one minor problem. Since the analytic formula is only approximate, the analytic phase may sometimes fail to detect a solution. This can happen when a solution exists near the reach boundary of the actual mechanism but may lie outside the workspace of the analytic approximation. This problem can be mitigated by the following heuristic. If a solution cannot be found at the fixed shoulder position or if the desired position requires a nearly fully extended elbow, we sample a variety of shoulder positions corresponding to different configurations of the clavicle joint. If the analytic phase succeeds for any of these configurations the numerical procedure is used to confirm if an actual solution exists. The simplest scheme for choosing the sampling points for the shoulder is to choose eight positions corresponding to the limits and midpoints of the clavicle joints as shown in the table below

Elevation	Abduction	Shoulder position relative to clavicle
0	-44	(-1.7,-2.64,11.0)
0	42	(1.62,-2.66,11.0)
0	128	(4.89,-1.40,10.2)
90	-44	(-1.43,-6.54,9.24)
90	42	(1.36,-6.57,9.23)
90	128	(4.513,-4.6,9.42)
179	-44	(-1.35,-7.24,8.72)
179	42	(1.29,-7.24,8.72)
179	128	(3.98,-6.72,8.31)

3.2 Method 2: A variation of Manocha and Canny's algorithm

I've also tried to find a more direct inverse kinematics solution for Jack's arm model. Manocha and Canny have recently developed a numerical method for 6R mechanisms that finds all solutions for a desired position and orientation. However, their algorithm is designed to work for independent joints expressed in Denavit-Hartenberg notation and is not readily amenable to the coupled joint mechanism used in Jack. In order to use their algorithm, it is necessary to find a set of three DH matrices equivalent to the five interdependent joints in Jack. More formally, we wish to find three Denavit-Hartenberg matrices A_1, A_2, A_3 satisfying

$$A_1(\theta)A_2(\phi)A_3(\tau) = R_y(\theta_1)R_x(\phi_1)T_1R_x(-\phi_1)R_z(\theta_2)R_x(\phi)R_z\left(\tau + \left(\frac{\phi}{90} - 1\right)\theta_2\right) \tag{14}$$

The first question is whether or not such a parameterization is even possible. In fact, any coupled system of joints with only n degrees of freedom can be represented by an equivalent "virtual" system of n independent joints, but this representation is valid only instantaneously. Thus, unlike a physically realizable device, the n independent joints are not fixed but change with the configuration of the mechanism. Mathematically this means that the Denavit-Hartenberg parameters $(\alpha_i, d_i, a_i\ i = 1..3)$on the left hand side of equation 14 are not constants but functions of θ, ϕ,and τ. I have attempted to solve 14 for a suitable set of values for the unknowns $\alpha_1, \alpha_2, \alpha_3, a_1, a_2, a_3, d_1, d_2, d_3$ as functions of θ, ϕ,and τ. However, the result-

ing equations are very ugly and I have not yet found a satisfactory solution. Moreover, the algorithm by Manocha and Canny relies on the fact that the joint variables are isolated in each matrix, which will no longer be the case for our "virtual" joints. Despite these problems, I am not yet convinced that this scheme is infeasible and I will continue pursuing it.

4 Exploiting Redundancy

In the previous sections, we assumed that one of the wrist joints was fixed at a constant angle. In practice, it may not always be straightforward to determine which joint to hold constant or to decide upon a suitable angle for the stationary joint. If a poor decision is made, the inverse kinematics solution may produce an "awkward" looking wrist posture. An even more serious problem is that the simplification of the arm to a six degree of freedom system reduces the reachable workspace which may cause the inverse kinematics procedure to fail even when a solution exists. Thus, it seems that a better scheme would be to exploit the extra degree of redundancy and to choose the solution that best satisfies an additional optimization criterion. For example, there is empirical evidence that humans tend to try to minimize wrist torques and one way of approximating this behavior would be to choose the solution that minimizes the displacement of the wrist angles from their rest position. I am currently investigating both analytical and numerical methods that utilize redundancy in the hope that these techniques will yield more "natural" looking postures for human arm inverse kinematics.

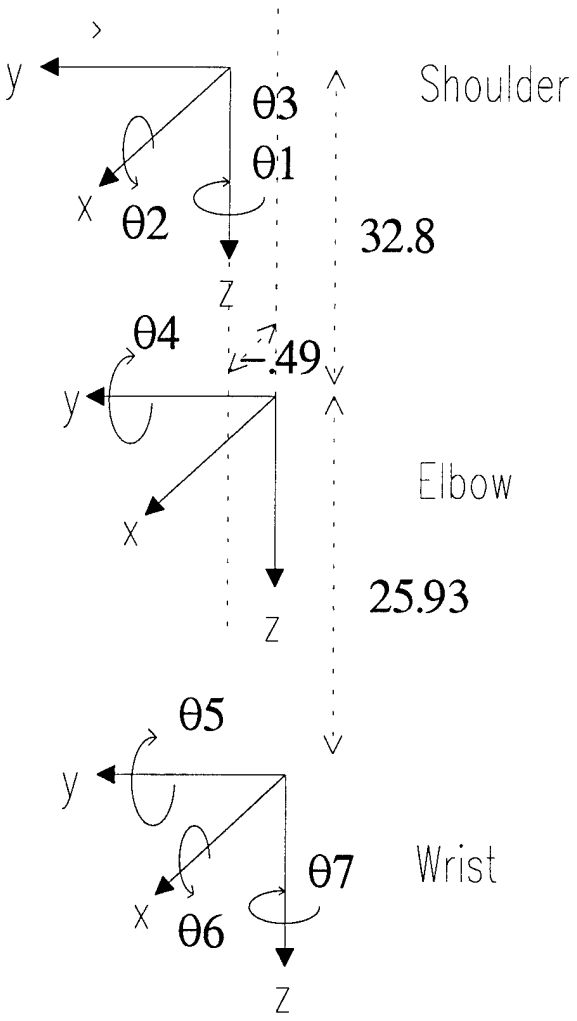


Figure 1: Peabody representation of the human arm

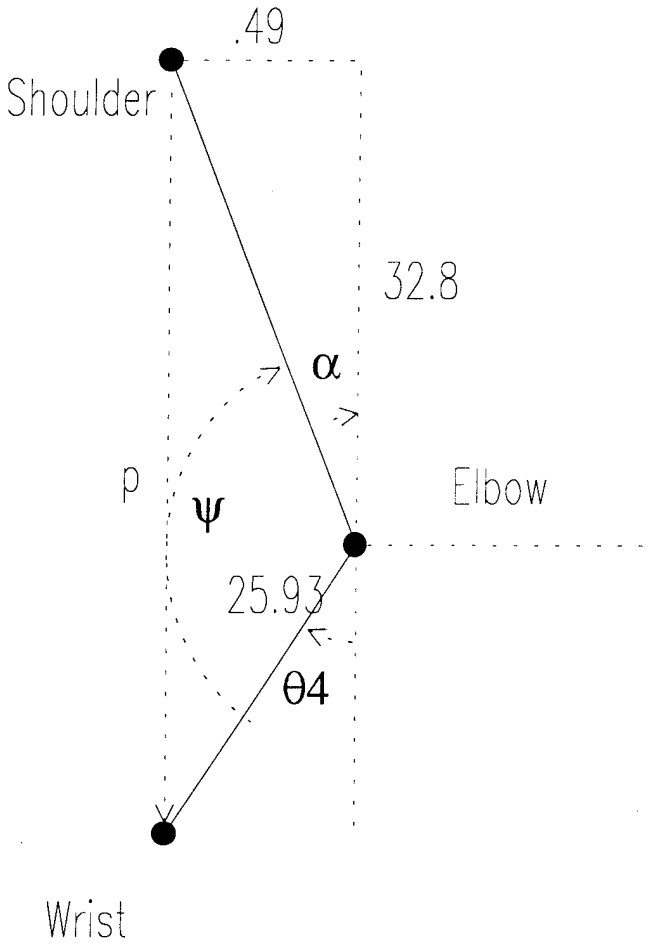


Figure 2: Calculating the elbow angle

B Adaptive Deformable Model Evolution Using Blending: DeCarlo and Metaxas

Adaptive Deformable Model Evolution Using Blending

Douglas DeCarlo and Dimitri Metaxas
Department of Computer & Information Science
University of Pennsylvania
Philadelphia PA 19104-6389
dmd@gradient.cis.upenn.edu, dnm@central.cis.upenn.edu

Abstract

We propose a new paradigm for shape representation and estimation which represents an object using a hierarchy of physics-based blended deformable components. Initially, the model is a sphere. In a physics-based way, the models deform and are replaced by more complex models allowing the evolution from the initial shape to the final shape. This evolution into more complex models is based on the arbitrary blending of shapes. Through hierarchical blending, multiple blends of the initial shape can occur. Since these models are defined based on global parameters, the shape recovered is a natural symbolic description. The descriptive power of these models allows the representation of objects with multiple holes. We present experiments involving the extraction of complex shapes, including an example of an object with multiple holes.

Keywords: Shape and Object Representation, Physics-Based Modeling, Shape Blending, Shape Estimation

Adaptive Deformable Model Evolution Using Blending

Abstract

We propose a new paradigm for shape representation and estimation which represents an object using a hierarchy of physics-based blended deformable components. Initially, the model is a sphere. In a physics-based way, the models deform and are replaced by more complex models allowing the evolution from the initial shape to the final shape. This evolution into more complex models is based on the arbitrary blending of shapes. Through hierarchical blending, multiple blends of the initial shape can occur. Since these models are defined based on global parameters, the shape recovered is a natural symbolic description. The descriptive power of these models allows the representation of objects with multiple holes. We present experiments involving the extraction of complex shapes, including an example of an object with multiple holes.

Keywords: Shape and Object Representation, Physics-Based Modeling, Shape Blending, Shape Estimation

Summary

(1) What is the original contribution of this work?

We develop a new physics-based modeling approach to shape estimation where the initial model parameterization is that of sphere. Based on the forces exerted by the data, the model can change its representation adaptively through blending. Therefore, with no a priori parameterization, the model evolves and locally re-parameterizes in a physics-based way. In addition, through this technique we can represent models which include multiple holes. Finally, the resulting parameterization is based on global shape, allowing the symbolic description of the recovered shape.

(2) Why should this contribution be considered important?

A shape representation should cover the widest possible variety of shapes which can be formed form a single model. Our approach achieves this goal. Furthermore, it can represent in a compact way more complex objects than any other technique known to us. Our technique is useful in reconstruction and should have applications in recognition.

(3) What is the most closely related work by others and how does this work differ?

While there have been several other global models (such as CAD based models) that can represent shapes with multiple holes, the shapes are not represented in a unified and compact way (see introduction for more details).

(4) How can other researchers make use of the results of this work?

These new models and estimation techniques can be applied to problems where shape modeling and characterization with a few parameters is important. i.e., reconstruction, recognition, and modeling.

1 Introduction

A challenging open problem in shape representation and estimation is the development of models that can be used in applications ranging from reconstruction to recognition. The difficulty in creating such models is the often conflicting requirements of reconstruction and recognition, i.e., representational accuracy versus symbolic descriptive power. As a result, most researchers have addressed this problem by creating models that employ a lot of parameters and are appropriate for reconstruction only [5, 14, 15, 25, 27] or have developed models with few descriptive parameters suitable for recognition tasks [1, 2, 3, 8, 9, 17, 21, 24].

Recently, models designed for application in both shape reconstruction and shape recognition have been presented [4, 6, 7, 13, 19, 22, 28]. Both [22] and [28] provide methods where the collection of parameters is ordered by level of detail. Techniques based on superquadrics [6, 7, 13] were presented to obtain part-level models. The models in [11, 19, 26] incorporate global deformations which represent prominent shape features, and local deformations which capture surface detail. For these models, correct configurations are extracted using a physics-based framework where forces are exerted by the data. In [4] a new class of deformable models based on axial shape blending was proposed. The distinguishing feature of blended models compared to all previous shape models is that in addition to being able to represent more complex objects in a compact way, they also have the additional power of *compactly* representing objects with varying topology, such as a sphere and a torus.

Regardless of how general or specific the currently existing shape models are, they are all limited by the assumption of a predefined parameterization that is determined by the underlying *predefined* geometric or physical properties of the model used. For example, in

modal analysis [22] the modal calculation depends on some predefined elastic properties. In in the models defined in [4, 19] predefined types of local (e.g., thin plate or membrane) and global (e.g., tapering, bending) deformations constrain the classes of objects that can be represented.

A truly general shape model should have no a priori parameterization and imposition of physical properties if it can be applied across a large number of objects. In the absence of prior knowledge, such a shape model should have an initial shape that is not favoring any spatial topology—therefore it should be a sphere. Furthermore, the associated shape estimation technique should allow the model to evolve and change parameterization and properties based on the shape of the given data. In other words the parameterization should be data driven and not predefined. Finally, the object representation should be able to satisfy as discussed above the requirements of both reconstruction and recognition.

In this paper we propose a new paradigm for shape representation and estimation which uses a new class of physics-based deformable models. Starting from the initial model of a sphere, the models evolve based on data forces. The representation of the model is changed to reflect the shape of the data. This local adaptation of global shape is based on blending of arbitrarily shape and is a generalization of the axial blending presented in [4].

The shape evolution, based on the localized blending, also allows the adaptation of the topology of the model. This amounts to being able to represent objects with no holes, as well as objects with multiple holes oriented arbitrarily.

Based on the underlying shape of the data, multiple local re-parameterizations of the global parameters may be necessary. In our technique this is done through hierarchical

blending. The resulting model representation is a blend of a series of globally parameterized shapes allowing in addition to shape accuracy, a symbolic shape description. Such a symbolic shape representation can be expressed as a tree where the leaves are the parameterized primitives and where inside-outside relationships are described. Therefore, as opposed to all previous part-based shape estimation techniques, our model can be represented as a set of connected components where each component is an integral part of the model. Fig. 2 shows the result of blending of two primitives. An additional benefit of using the above models which are parameterized based on global deformations is their robustness to noise and their suitability in estimating the underlying shape of sparse and incomplete data, without imposing any prior smoothness.

In this paper, we first define the geometry of the new class of deformable models which is based on the hierarchical blending of arbitrarily oriented shapes with global deformations. Then we present how blended models can be incorporated into the previously developed physics-based estimation framework presented in [19, 26]. We then present modifications to this framework that are necessary in order to be able to dynamically evolve an initial shape through blending. Finally, we demonstrate our technique through a series of experiments involving incomplete range data from various objects with varying topologies.

2 Geometry of blended shapes

We will be incorporating blended shapes into the physics-based deformable model framework introduced in [19, 26]. In the following sections, we will describe the process of shape blending using many example shapes. In section 4.1, we will see how our vision system starts with

a model of a sphere, and incrementally blends where necessary based on forces from the data. This adaptive blending process will cause an evolution from a sphere model to the final model which represents the data.

While local deformations can be added on top of our blended shapes to capture shape detail, this will be defeating the purpose of our adaptive blending scheme. Even small bumps can be represented using blending–although it will most likely use a number of parameters on the order of a local representation. If local deformations are desired, then we can define a minimum feature size that can be blended. Any smaller features (such as small bumps) can be captured by local deformations.

2.1 Deformable model geometry

The models used in this paper are 3-D surface shape models. The position of a point on the model is given in world coordinates by $\mathbf{x}$ which is the result of a translation and rotation of its position $\mathbf{s}$, with respect to a fixed (non-inertial) reference frame. The material coordinates $\mathbf{u} = (u, v)$ of these shapes are specified over a domain Ω. The position of a point on the world model at time t, with material coordinates $\mathbf{u}$, with respect to an inertial frame of reference is

$$\mathbf{x}(\mathbf{u}, t) = \mathbf{c}(t) + \mathbf{R}(t)\mathbf{s}(\mathbf{u}, t), \tag{1}$$

where $\mathbf{c}$ is the center of the inertial frame, and $\mathbf{R}$ is a rotation matrix which specifies the relative orientation of the inertial frame to a fixed reference frame.

In the fixed reference frame, the position of model points is defined by a reference shape $\mathbf{s}$. This reference shape $\mathbf{s}$ is constructed by applying a global deformation $\mathbf{T}$ (such as bending)

with parameters $\mathbf{q_T}$ to a shape primitive $\mathbf{e}$ as follows:

$$\mathbf{s(u)} = \mathbf{T(e; q_T)}. \tag{2}$$

$\mathbf{T}$ can be a composite sequence of primitive deformation functions, $\mathbf{T(e)} = \mathbf{T}_n(\mathbf{T}_{n-1}(\ldots \mathbf{T}_1(\mathbf{e})))$.

For every 3-D shape primitive (such as a superellipsoid [1]), we have $\mathbf{e} : \Omega \rightarrow \mathbb{R}^3$. An example primitive (in this case, a sphere) is shown in Fig. 1, and shows how the material coordinates (the domain Ω) are "folded up" resulting in the closed shape $\mathbf{e}$.

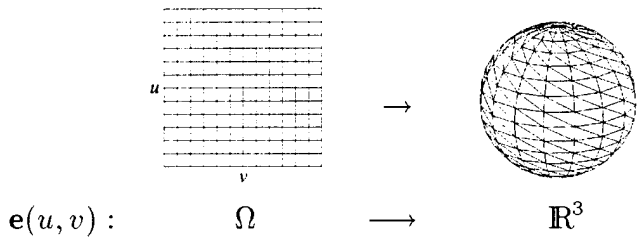


Figure 1: Example shape primitive function $\mathbf{e(u)} = \mathbf{e}(u, v)$

For a superellipsoid, $\Omega = [-\pi/2, \pi/2] \times (-\pi, \pi]$. When folding up this space, we first make a "tube" by identifying $-\pi$ and π in v (which amounts to connecting the left and right sides of Ω in the diagram). We can then form the poles by identifying all points with values of $u = \pi/2$ together (for the north pole), and all points with values $u = -\pi/2$ together (for the south pole), which in effect closes each end of the tube.

To represent the geometry of the primitive, a mesh of nodes is used, where each node is assigned a unique point in Ω. The edges connecting the nodes represent connectivity of the nodes in Ω space. Nodes can be merged together to form a closed mesh where points in Ω map to the same 3-D model location (such as for the poles of a sphere). As a result, the topology of the mesh agrees with the topology of the shape primitive.

For the applications in this paper, we will be using the superellipsoid and supertoroid primitives [1], although any other parameterized primitives could be used. We will later see how our vision system starts with a sphere model and uses these components to build a blended shape. In the next section, we describe how by combining together these shape primitives, we produce a shape model with much greater shape coverage.

2.2 Blended shape geometry

A blended shape is a combination of two component shapes. The desired pieces of each component shape are selected, and are "glued" together. This gluing is performed using linear interpolation [4], which produces a continuous blended shape as a result. We can blend together two shapes using the following formula:

$$\mathbf{s}(\mathbf{u}) = \mathbf{s}_1(\mathbf{u})\alpha(\mathbf{u}) + \mathcal{R}\Big(\mathbf{s}_2(\mathcal{B}(\mathbf{u}))\Big)\Big(1 - \alpha(\mathcal{B}(\mathbf{u}))\Big), \tag{3}$$

where $\mathbf{s}_1$ and $\mathbf{s}_2$ are the component shapes, as in Fig. 2(a), $\alpha : \Omega \rightarrow [0,1]$ is the blending function which controls how each of the component shapes are expressed and how they are glued together. α is specified using the blending regions shown in Fig. 2(b). These blending regions are overlaid on the material coordinate spaces of the component shapes. The portion of these spaces corresponding to the retained surface is shown in white, while the portion which is removed is shown in gray. The retained portions of the surfaces are shown in Fig. 2(c). Basically, the blending regions specify where to "cut" the shape (the boundary line), and which part of the shape to keep (white or gray). Ideas similar to these cutting and gluing operations have been used in studying manifold surgery for topology [10].

Manifold surgery has been used to construct manifolds of arbitrary topology using a few simple components. $\mathcal{B} : \Omega \rightarrow \Omega$ is an invertible function which maps the domains of the selected region of $\mathbf{s}_2$ into the region which was removed from $\mathbf{s}_1$. $\mathcal{R} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is a rigid transformation (a rotation and a translation), which aligns $\mathbf{s}_2$ with $\mathbf{s}_1$. For the moment, we will assume that both $\mathcal{B}$ and $\mathcal{R}$ are identity functions. Finally, $\mathbf{s}$ is the resulting blended shape, shown in Fig. 2(d).

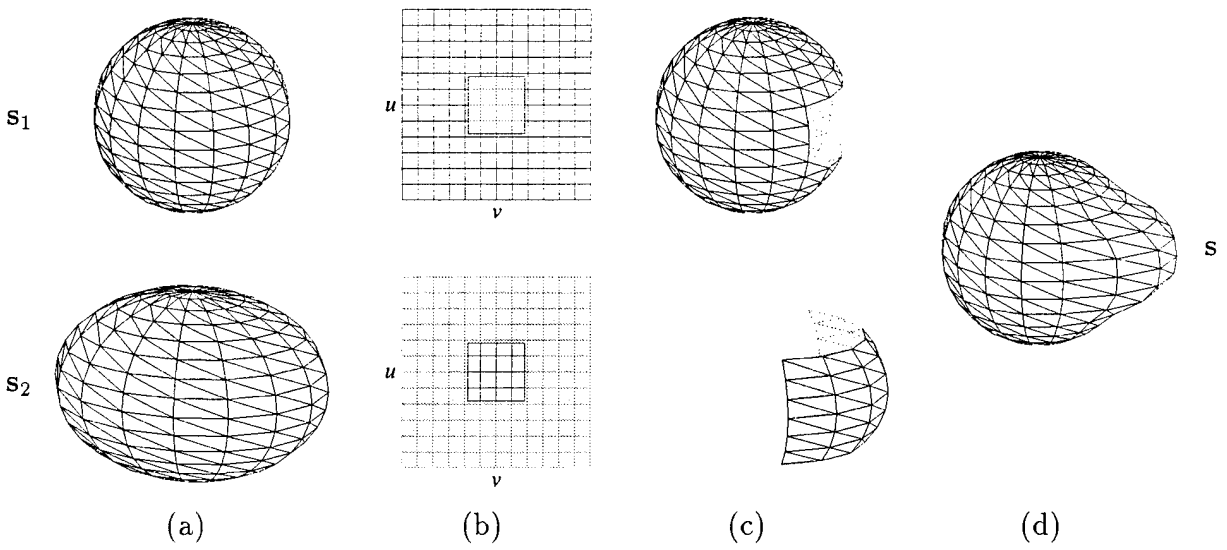
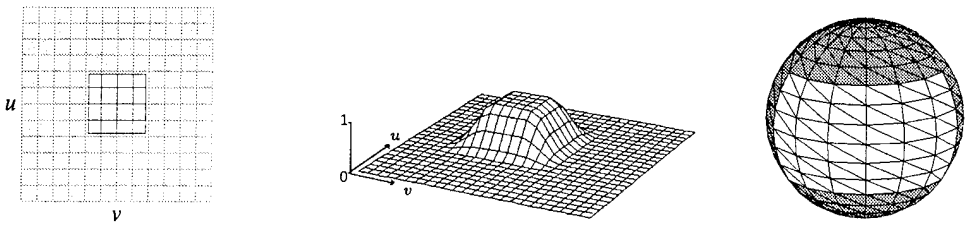


Figure 2: (a) Shape primitives $\mathbf{s}_1$ and $\mathbf{s}_2$ (b) Blending regions (c) Restriction of shapes to blending regions (d) Resulting blended shape $\mathbf{s}$

The resulting blended shape $\mathbf{s}$ shown in Fig. 2(d) has a smooth transition between each of its components. This will be the case if α has at least C^1 continuity, as shown in Fig. 3. For the example blending region given in Fig. 3(a), the resulting blending function α is shown in (b). Notice how α is 1 where the region is white, and 0 where the region is gray. A smooth transition from 0 to 1 connects these two regions. The area on the shape primitive affected by this blending region is displayed in Fig. 3(c) as the white region on the shape. A discussion of the implementation of α, as well as the size of the transition region (the

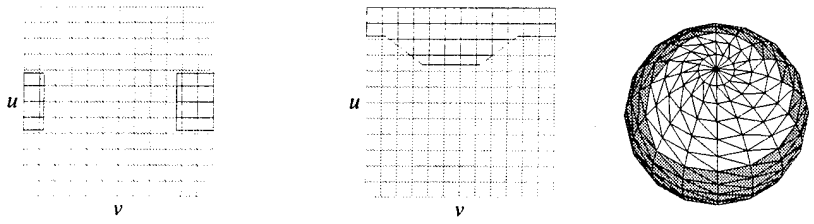
"steepness" of the plateau) is given in section 2.3.



(a) (b) (c)

Figure 3: (a) Example blending region (b) Blending function α corresponding to this region (c) Area on shape corresponding to this region

When defining the boundary of blending regions, we must consider the topology of the underlying shape. Fig. 4(a) shows a single blending region which includes part of the "seam of the tube" where $v = \pi$ and $v = -\pi$ meet. Fig. 4(b) shows an example blending region which includes the north pole of a sphere, as well as the area on the sphere which is affected by this region. The representation of blending regions is described in section 2.3.



(a) (b)

Figure 4: (a) A blending region which "wraps around" in Ω along v (b) A blending region which includes the north pole (also shown on the shape)

In Fig. 2(b), the blending regions of each of the component shapes lines up exactly, so that $\mathcal{B}$ is the identity function. Usually the blending region boundary for $\mathbf{s}_2$ does not have such a simple correspondence with the blending region boundary for $\mathbf{s}_1$. Fig. 5(b) shows how $\mathcal{B}$ maps the blending region for $\mathbf{s}_2$ to allow a simple correspondence with $\mathbf{s}_1$, shown in (a). Note that $\mathcal{B}$ maps both the boundary and the space surrounding it. This permits a

correspondence to be performed for those points off the boundary where linear interpolation is performed (where $0 < \alpha < 1$). One must be careful in preserving the orientation of the blending boundaries. Each boundary has an implicit direction associated with it (clockwise or counter-clockwise) which must be preserved by $\mathcal{B}$. Not doing so can produce a surface with a self-intersection where it is turned inside-out.

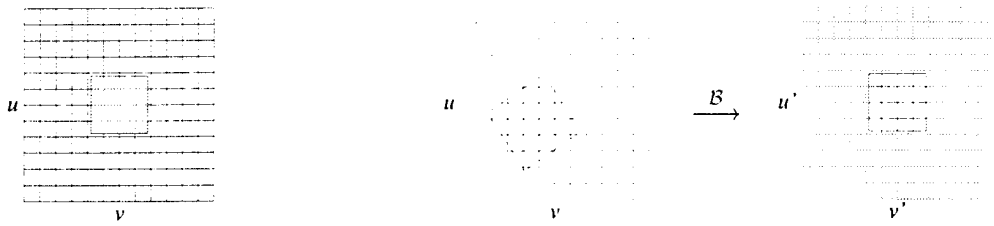


Figure 5: (a) Blending region for $\mathbf{s}_1$ (b) Blending region for $\mathbf{s}_2$ (c) Blending region for $\mathbf{s}_2$ after being mapped by $\mathcal{B}$

In addition to the correspondence in material coordinate space performed by $\mathcal{B}$, there must also be a correspondence in 3-D space. In other words, the two components must be "lined up" before blending is performed. This spatial correspondence is performed by applying a rigid transformation (translation and rotation) to the primitive $\mathbf{s}_2$. An example of this spatial alignment is shown in figure Fig. 6, where the rigid transformation is shown in Fig. 6(b) to (c). The blending regions used by this shape are given in Fig. 5(a) and (b).

The mesh of the blended result in Fig. 6(d) is a composite mesh formed by combining the meshes in (c). A mesh merging algorithm, such as the one presented in [23] can be used to perform this merging as a post-processing step to blending. The bottleneck in mesh merging algorithms is the nearest-node computations. In this case, the nearest-node computations become constant time operations since $\mathcal{B}$ provides the correspondence across meshes, and only small neighborhoods need to be checked to find the nearest node.

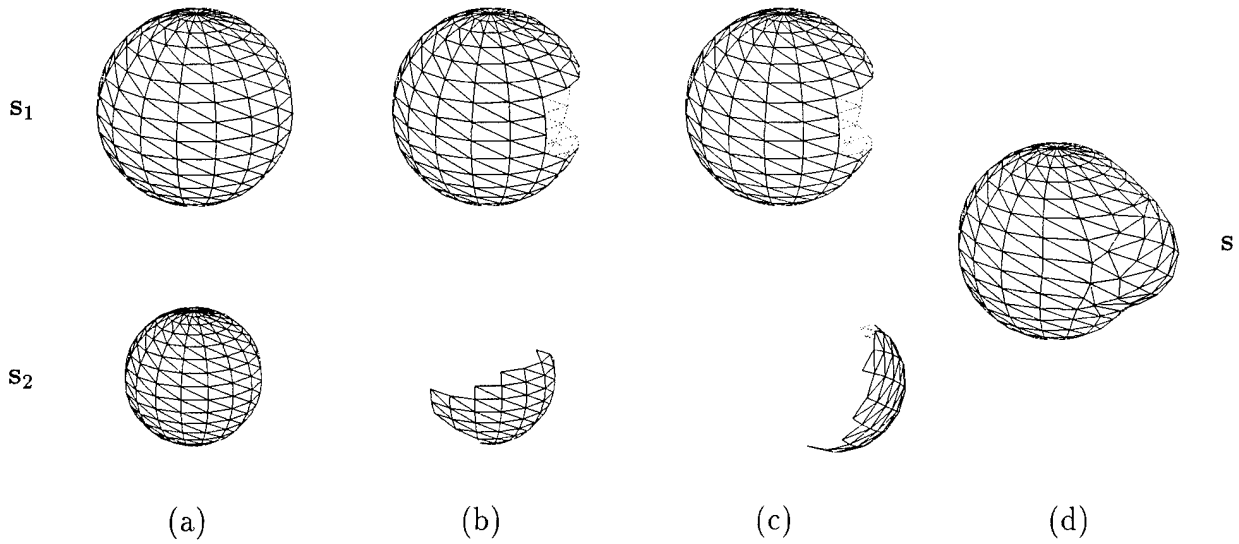


Figure 6: (a) Shape primitive s_1 and s_2 (b) Restriction of shapes to blending regions (c) Component shapes after rigid transformation of s_2 (d) Resulting blended shape

2.3 Representation of blending regions

For the applications in this paper, the blending regions are defined as rectangular regions in Ω space. Each region has 6 parameters, c_u, c_v, p_{1_u}, p_{1_v}, p_{2_u} and p_{2_v} (where the vector p_1 is linearly independent of p_2). There are two different representations used for blending regions, depending on their location in Ω—those which do not include the pole, shown in Fig. 7(a), and those which include the north pole, shown in Fig. 7(b). Blending regions which include the south pole are represented in an analogous manner.

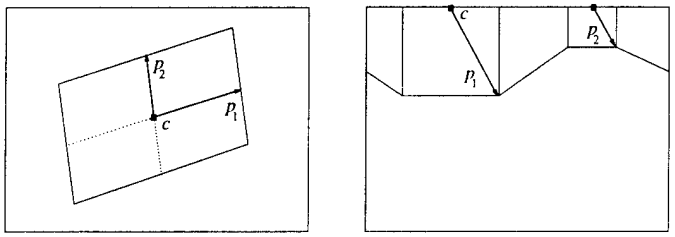


Figure 7: (a) Non-polar blending region (b) Blending around north pole

There are also additional parameters for each blended shape–a translation and rotation

to specify $\mathcal{R}$ (6 parameters), the parameter d which controls the extent of the transition region of α, and h which restricts the range of the blending function to $[h, 1]$. Normally, $h = 0$, but this parameter will be used to lessen or remove the effect of $\mathbf{s}_2$ on the resulting shape during hole addition (when $h = 1$, $\mathbf{s} = \mathbf{s}_1$).

A straightforward basis transformation operation takes points in Ω to points using the basis (p_1, p_2). We will denote the point $\mathbf{u} \in \Omega$ expressed in this basis as $\bar{\mathbf{u}}$. Points on the boundary of the blending region have $\|\bar{\mathbf{u}}\| = 1$, and points on the interior have $\|\bar{\mathbf{u}}\| < 1$. We can use $\bar{\mathbf{u}}$ to construct $\mathcal{B}$ by converting a value of $\mathbf{u}$ to $\bar{\mathbf{u}}$ using the blending region for one shape component, and then back using the blending region for the other component. Note that we must preserve the "handedness" of this coordinate system so that surface orientation is preserved by the blend.

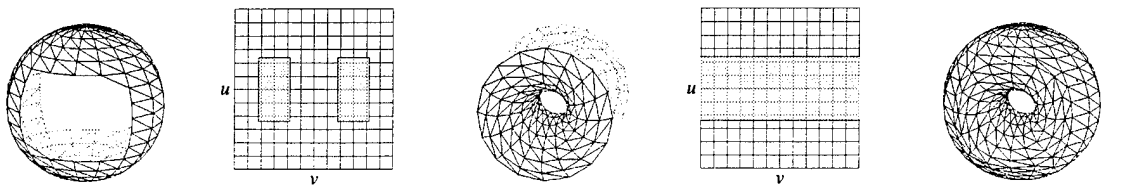
We can compute α from $\bar{\mathbf{u}}$ by setting α to 0 where $\|\bar{\mathbf{u}}\| < 1 - d$, to 1 where $\|\bar{\mathbf{u}}\| > 1 + d$, and to an intermediate value otherwise. Hence we see that d controls the "steepness" of α as seen in Fig. 3. To make α smooth, we can construct a piecewise polynomial surface, where a C^1 smooth connection between 0 and 1 is provided using a surface formed using the Hermite polynomial $H(x) = 3x^2 - 2x^3$ for $x \in [0, 1]$.

3 Topology and coverage of blended shapes

3.1 Hole additions

The addition of a hole requires two blending regions—two parts of a shape are cut out, and the hole is glued into this location. Fig. 8(a) shows a sphere with two parts removed.

The blended regions for this sphere are also displayed, showing how two regions in Ω are removed. Fig. 8(b) shows the hole of a torus. Two separate cuts are made on the torus, and the resulting hole is glued into the sphere, shown in Fig. 8(c).



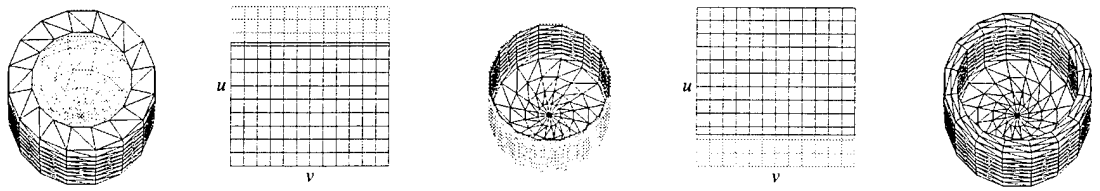
(a) (b) (c)

Figure 8: (a) Superellipsoid (s_1) (b) Supertoroid hole (s_2) (c) Blended result–a torus

Unlike the holes in [4] which could only be added between the poles of a sphere, the holes presented here are general, and can be added between any two locations of a blended object. We will see in section 4.1 how the hole additions are performed by the estimation system.

3.2 Indentations

Indentations actually do not require any additional machinery to be produced. But since many global shape representations fail to produce shapes with indentations or cavities, it is presented here. Fig. 9(a) shows a cylinder with part of the top removed by the given blending region. Fig. 9(b) shows a shape which is turned inside out by having a negative major axis length (in this case, $a_3 < 0$). This produces a closed surface where the surface is on the inside of the shape. Cutting off part of this surface reveals the surface on the inside, which can be used for making a cavity. When this cylinder and inverted cylinder are blended together, they produce the "cup" object seen in Fig. 9(c).



(a) (b) (c)

Figure 9: (a) Cylinder (s_1) (b) Inverted cylinder (s_2) (c) Blended result–a cup

3.3 Hierarchy of blended shapes

In the sections above, the blending described is performed on shape primitives only. Without any additional machinery, we can perform blending on two component shapes which are already blended. This results in a tree structure, where the leaves of the tree are shape primitives, and the internal nodes of the tree are blended shapes. Since $\mathcal{B}$ is invertible, we can use this mapping to find corresponding points on any two component shapes anywhere in this tree structure.

Since holes can be added anywhere, this means that an object of any topology can be potentially represented. Later, we will see an example of the fitting of a two-holed object in section 5.

We can produce a symbolic description of a blended shape which displays this hierarchy. For each of the experiments performed in section 5, a symbolic description is provided. Such a description may be potentially useful for recognition purposes.

4 Dynamics and generalized forces

In [4], the dynamics framework of [19] was extended to accommodate blending. We find that similar modifications are needed to incorporate the blending described here. In this frame-

work all the degrees of freedom of the model (translation, rotation, and global parameters) are collected together to form the generalized coordinates of the model, $\mathbf{q}$,

$$\mathbf{q} = (\mathbf{q_c}^\top, \mathbf{q}_\theta^\top, \mathbf{q_s}^\top)^\top, \tag{4}$$

where $\mathbf{q_c} = \mathbf{c}(t)$, $\mathbf{q}_\theta$ is the quaternion used to specify $\mathbf{R}(t)$, and $\mathbf{q_s}$ is given by

$$\mathbf{q_s} = (\mathbf{q_{s_1}}^\top, \mathbf{q_{s_2}}^\top, \mathbf{q}_b^\top, \mathbf{q_T}^\top)^\top. \tag{5}$$

$\mathbf{q_{s_1}}$ and $\mathbf{q_{s_2}}$ are the parameters of each of the component shapes, $\mathbf{q}_b$ are the parameters that specify α, $\mathcal{B}$ and $\mathcal{R}$, and $\mathbf{q_T}$ are the parameters of the global parameterized deformations. For a hierarchical blended shape, each of the component shapes $\mathbf{q_{s_1}}$ and $\mathbf{q_{s_2}}$ can either be a primitive, or child blended shape which will have the form given by (5).

When fitting the model to data, the goal of shape reconstruction is to recover the parameters in $\mathbf{q}$. The approach used here performs the fitting in a physics-based way—the data apply forces to the surface of the model, deforming it into the shape represented by the data [26]. In shape estimation applications, we use a simplified form of the Lagrange equations of motion [18] where the mass is set to zero (so that the model has no inertia and comes to rest as soon as the applied forces equilibrate or vanish):

$$\mathbf{D}\dot{\mathbf{q}} = \mathbf{f_q} = (\mathbf{f_c}^\top, \mathbf{f}_\theta^\top, \mathbf{f_s}^\top)^\top, \tag{6}$$

where $\mathbf{D}$ is the damping matrix and where $\mathbf{f_q}$ are the generalized forces [19]. These generalized forces can be further broken down into components each corresponding to a component

of $\mathbf{q}$ as given in (4) above.

Using (6), $\dot{\mathbf{q}}$ can be computed, and an integration method can be used to update $\mathbf{q}$. Performing this process iteratively results in the model more closely representing the desired shape. In this implementation, an adaptive Euler integration method is used to update $\mathbf{q}$.

We compute the generalized forces $\mathbf{f_q}$ from the 3-D applied forces. The computation of $\mathbf{f_c}$ and $\mathbf{f}_\theta$ are the same as described in [19]. The computation of $\mathbf{f_s}$ is given by

$$\mathbf{f_s} = (\mathbf{R}\mathbf{J_s})^\top \mathbf{f}_{\text{applied}}. \tag{7}$$

We compute $\mathbf{J_s}$, the Jacobian for the global shape $\mathbf{s}$, as follows:

$$\mathbf{J_s} = \partial \mathbf{s} / \partial \mathbf{q_s}. \tag{8}$$

The Jacobian of the global shape, $\mathbf{J_s}$, "converts" the applied forces into generalized forces, which will deform the global shape. The addition of blending changes the computation of $\mathbf{J_s}$. In particular, from (3) and (8):

$$\mathbf{J_s} = \left[\alpha(\mathbf{u})\mathbf{J}_{\mathbf{s}_1} \;\middle|\; \Big(1 - \alpha(\mathcal{B}(\mathbf{u}))\Big)\mathcal{R}_{\text{rot}}(\mathbf{J}_{\mathbf{s}_2}) \;\middle|\; \mathbf{J}_b \right], \tag{9}$$

where $\mathbf{J}_{\mathbf{s}_1} = \partial \mathbf{s}_1 / \partial \mathbf{q}_{\mathbf{s}_1}$ is the Jacobian for the first shape, $\mathbf{J}_{\mathbf{s}_2} = \partial \mathbf{s}_2 / \partial \mathbf{q}_{\mathbf{s}_2}$ is the Jacobian for the second shape, and $\mathbf{J}_b$ is the Jacobian for the parameters of the blending function, and reflects how the global shape $\mathbf{s}$ changes with respect to the blending function parameters $\mathbf{q}_b$.

$\mathcal{R}_{\text{rot}}$ is the rotational component of $\mathcal{R}$. From (3), we find that

$$\mathbf{J}_b = \frac{\partial \mathbf{s}(\mathbf{u})}{\partial \mathbf{q}_b} = \mathbf{s}_1(\mathbf{u}) \frac{\partial \alpha(\mathbf{u})}{\partial \mathbf{q}_b} - \mathcal{R}\Big(\mathbf{s}_2(\mathcal{B}(\mathbf{u}))\Big) \frac{\partial \alpha(\mathcal{B}(\mathbf{u}))}{\partial \mathbf{q}_b} \tag{10}$$

Given the implementation of α presented in section 2.3, we can construct a piecewise derivative of each of the blending parameters in $\mathbf{q}_b$.

4.1 Model evolution

Our initial model in our estimation system is always a single superquadric primitive having the shape of a sphere. During the fitting process, we can decide to perform blending based on the shape and forces from the data points. When we split the model this way, we must define the blending regions, as well as specify the other shape we are blending with. The experiments in section 5 give examples of this model splitting, showing an evolution from a sphere to the final model.

For a given shape model and data, the shape estimation process eventually reaches a steady state, where all applied forces equilibrate or vanish. When these forces do not vanish, it is due to the inability of the model to reach a shape that fits the data well. Fig. 10(a) shows an example situation where forces are equilibrated. A superellipsoid cannot produce the tapered shape represented by this data.

We can apply a region growing algorithm [20] (where the region consists of nodes, and the connectivity is defined by the mesh) to find these regions on a shape where there are non-zero forces. Fig. 10(b) shows such a region for the shape and data given in (a).

During the estimation process, we can compute these regions on the shape where the

(a) (b)

Figure 10: (a) 2-D cross section of a superellipsoid with forces from data points that have equilibrated (b) region on shape where forces have equilibrated

forces have equilibrated. We can then replace the current model with a blended model. The boundary of both blending regions is set to the boundary we computed from the region growing (so initially $\mathcal{B}$ is the identity mapping). We also initially set all relevant parameters in $\mathbf{s}_2$ to those taken from $\mathbf{s}_1$ (for a superquadric, a_1, a_2, a_3, ϵ_1, ϵ_2).

This operation only changes the representation of the model, not its shape. The benefit of performing this split, is that in the area of this region of force equilibration, the shape can now deform to capture the deviation in the data that is causing the equilibration. Looking at (9), we notice that the blending function has the desirable effect of localizing the effect of a force to the appropriate shape component. So the forces that were pulling on $\mathbf{s}_1$ in the blending region will now cause the deformation of $\mathbf{s}_2$.

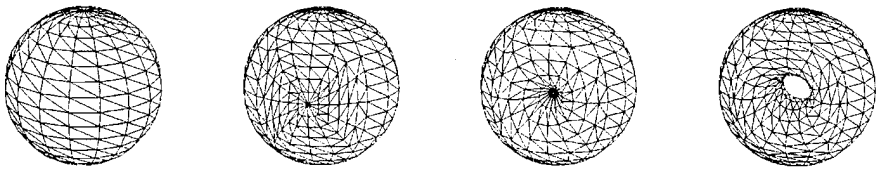
This method also applies to fitting shapes with indentations or cavities. If the size parameters (a_1, a_2 and a_3) of $\mathbf{s}_2$ become smaller (or even negative), then there will be an indentation present at that location.

The addition of a hole to the model involves the proximity detection of different parts of the model. Surface intersection detection methods such as [16] can be used. Once this proximity is detected, we can correctly position the hole and construct a blended shape. This involves specifying $\mathcal{R}$ so that the ends of the hole line up with the removed parts of

the original shape. Holes do not necessarily have to form between two locations on the same shape primitive. If the hole is deep, it is very common for the hole to form between two parts of the model which are cavities which formed due to the presence of the hole in the data. This can been seen in the experiment in Fig. 15(f)-(g). The blending regions for $\mathbf{s}_1$ are computed from the region growing computation described above. As described in section 3.1, the only part of the supertoroid that is used is the "hole". The initial blending region for $\mathbf{s}_2$ can be taken from Fig. 8(b). We must also specify $\mathcal{R}$, so the location of the hole is aligned with the regions on $\mathbf{s}_1$. This can be performed by a seperate fitting process where the 3-D positions of boundary regions of $\mathbf{s}_1$ can pull on a model containing the boundary regions of $\mathbf{s}_2$. A bending deformation can be added to the hole to include bent holes.

Fig. 11 shows the transformation from a shape model shown in (a) (in this case, a sphere) to include a hole (d). The model shown in Fig. 11(b) is the model which is constructed when the blend is created. The torus hole is not expressed in the resulting shape since $h = 1$, and the topology of the mesh of the torus is set to that of a sphere [4]. During the fitting process, if a hole is present in the data, data forces will change h to have the value 0, shown in Fig. 11(b). When $h = 0$, the topology of the torus mesh is set to be that of a torus, and the shape is a torus with a zero sized hole. When $h > 0$, then we do not permit the torus hole to open, since doing so will produce an open surface.

It is by these two processes–the addition of a blending region of the same shape type, and the addition of holes, which produce the evolution of the model from a sphere to the final model. Neither of these processes changed the shape of the model initially. Instead, they added blending to the model in a particular location which allowed the model to gain



(a) (b) (c) (d)

Figure 11: Hole addition (a) Starting shape (b) Shape blended with torus with $h = 1$ (c) Shape blended with torus with $h = 0$ (d) Shape blended with torus with open hole

a better fit on the data. Then, the parameters used to specify α, $\mathcal{B}$ and $\mathcal{R}$, in addition to the global parameters, are updated by the fitting process by applied forces from the data.

4.2 Force-model assignment

When fitting data which represent a shape with a cavity or hole, care must be taken in assigning forces from the data points. If a nearest-node assignment method is used, the points on the inside of a cavity or hole may pull on the back side of the surface nearby. The reason why this point choice is incorrect is because the data is pulling on a point behind the surface that the data represents. In other words, the range data does not distinguish the "inside" of its shape from its "outside".

When taking range data, it is important to record (at least roughly) where the sensor position was for each data point. This information is often omitted from range data sets. If we have a vector for each data point which indicates the direction of the sensor, we know that half-space which includes that vector also includes the normal to the surface of the scanned object. Therefore, when we choose which node of our model to attach to a data point, that model node should be in this half-space. This modification allows fitting of objects with cavities or holes. Fig. 15 is an experiment where this force-model assignment method

produces the correct fit.

5 Experiments

In the following three fitting experiments, we show the results of using our shape estimation system. Fig. 12 shows information on each of the experiments including the source and size of the data set, the number of parameters in the final model, the number of iterations taken by the solver, and the resulting mean squared error (MSE).

Data	Source	Points	MSE	#Parm	Iter
block/cylinder	MSU (`column2`)	1034	1.35%	29	190
mug	MSU (`cup1`)	1207	2.83%	68	372
two holed	CAD generated	893	1.02%	83	511

MSU: data from the Michigan State University PRIP database [12].

Figure 12: Experiment data and statistics

In each of the three examples, the data set and initial model of a sphere is shown. The first fit shown is a rough fit—by fitting the a_1, a_2 and a_3 parameters of the initial sphere.

Fig. 13(d) shows the first adaptation of the shape model, where a blending region was added (in gray). The blended region added is shown in (e). After fitting the new shape model, the shape in (f) is formed, followed by the final fit in (g), and the final blending regions in (h). A symbolic representation of the object is shown in (i).

The fitting of a mug is shown in Fig. 14. The blending region which corresponds to the mug handle forms in (d), and after rough fitting is shown in (e) with the corresponding blending region in (f). After further fitting of the handle in (g), a hole blend is added in (h) with blending region (i). After rough fitting and hole opening (j), the final fit is obtained

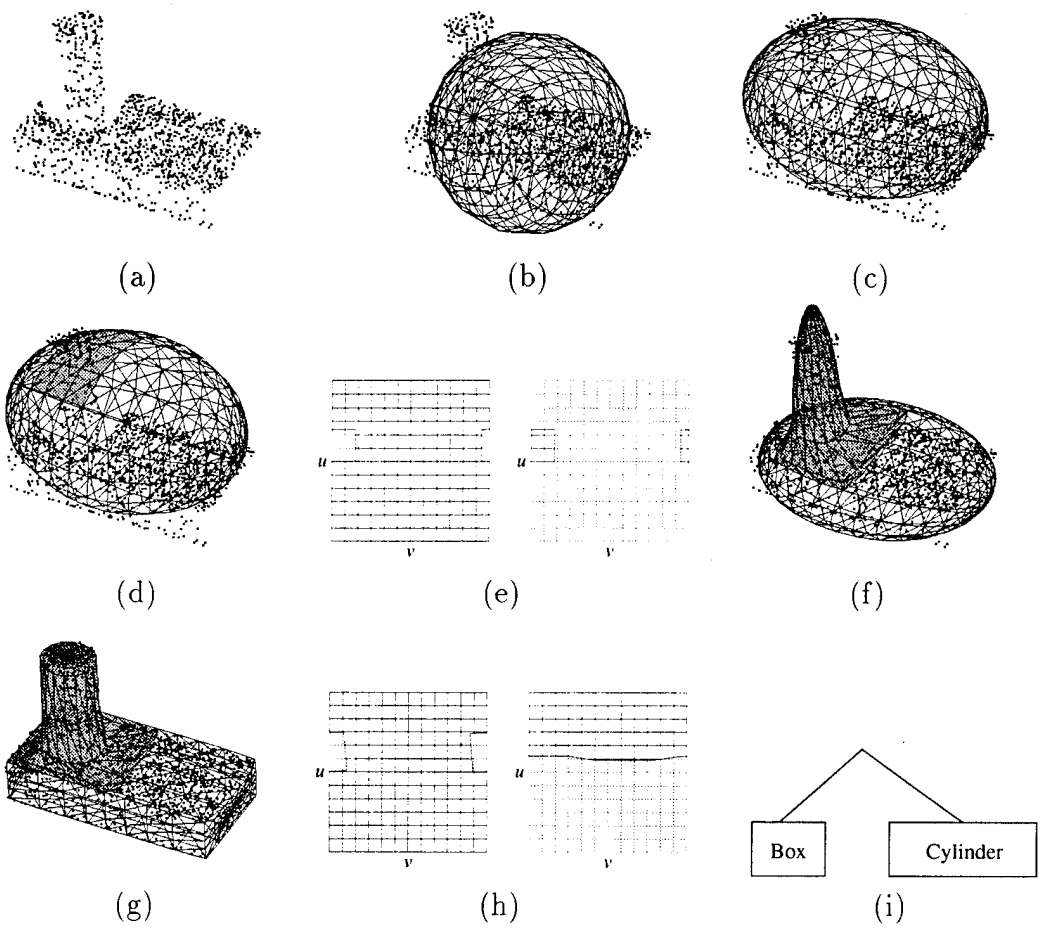


Figure 13: Fitting of a block and cylinder: (a) range data (b) initial model (c) model after rough fit (d) blending added (e) initial blending regions (f) rough fit after blending (g) final fit (h) final blending regions (i) symbolic representation

(k). A symbolic description of a mug is shown in (l).

Finally, the fitting of an object with two holes is shown in Fig. 15. A indentation blending region for one of the holes is formed in (d), and after initial fitting becomes indented (e). After several more blends, the shape evolves two additional indentations (f). After further fitting, a hole has formed in (g). The second hole has formed in (h), and the final fitted result is shown in (i).

6 Conclusion

We have presented a new approach to shape modeling where the models used are not pre-parameterized, and have the ability to adapt to the topology of the given data based on the forces exerted from the data to the model. The models automatically evolve, based on blending of parameterized shapes. The blending allows both accuracy in shape estimation and symbolic shape description. The techniques presented here also allow a hierarchical evolution of the initial spherical shape to fit the given data, making the estimation process more robust. Since the model is based on global parameterizations, the technique is robust to noise and can be used in sparse and incomplete data. Finally, unlike all other previous part-based shape estimation techniques, our shape representation is based on a connected component description where each component is an integral part of our model. We have presented shape estimation results for objects with varying topology whose shape could not be estimated compactly and robustly with previous shape models and techniques.

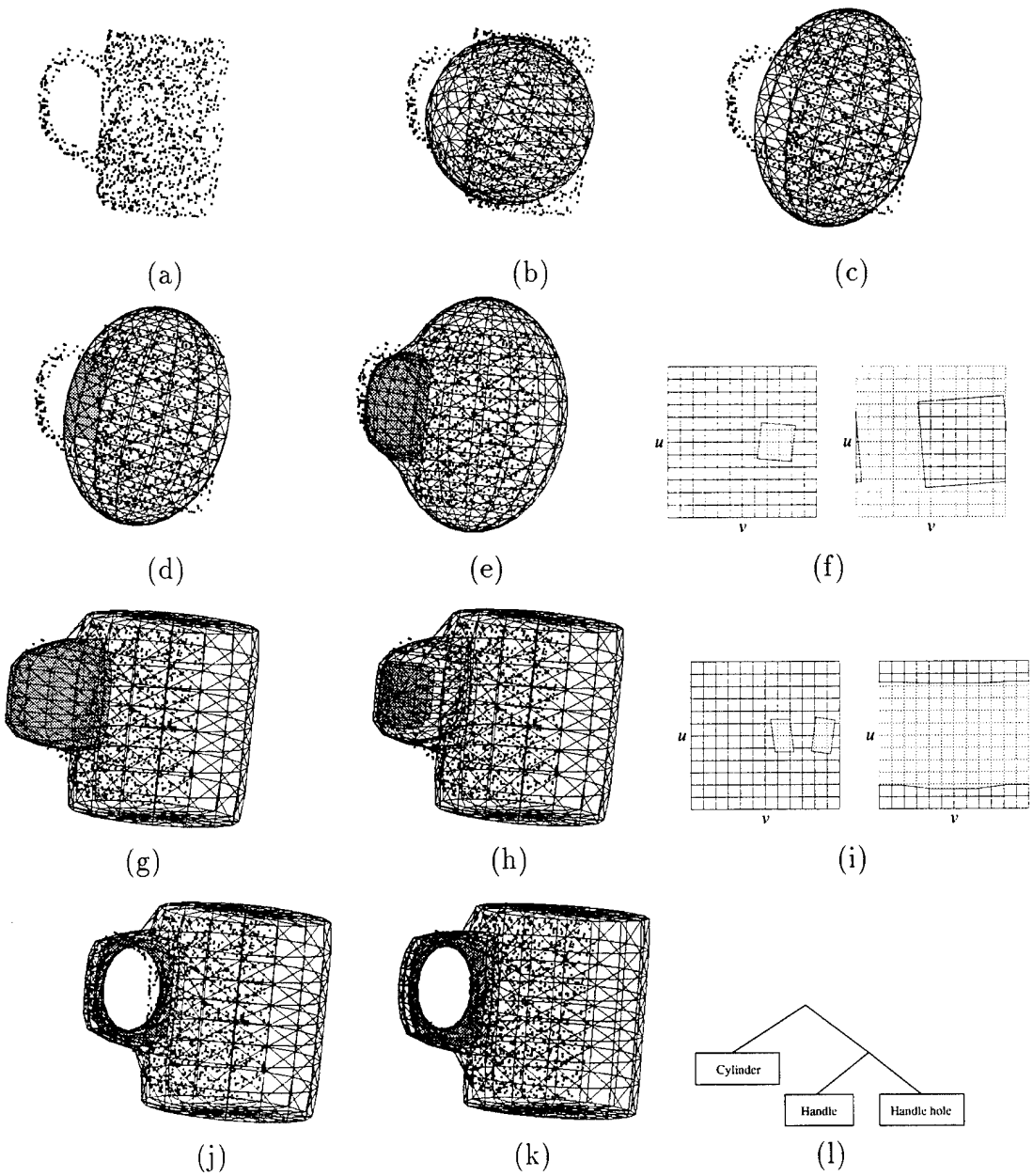


Figure 14: Fitting of a mug (a) range data (b) initial model (c) rough fit (d) blending region added (e) rough fit of blended shape (f) blending regions after rough fit (g) further fit of blended shape (h) addition of hole blending region (i) blending regions for hole (j) rough fit (k) final fit of mug (l) symbolic representation

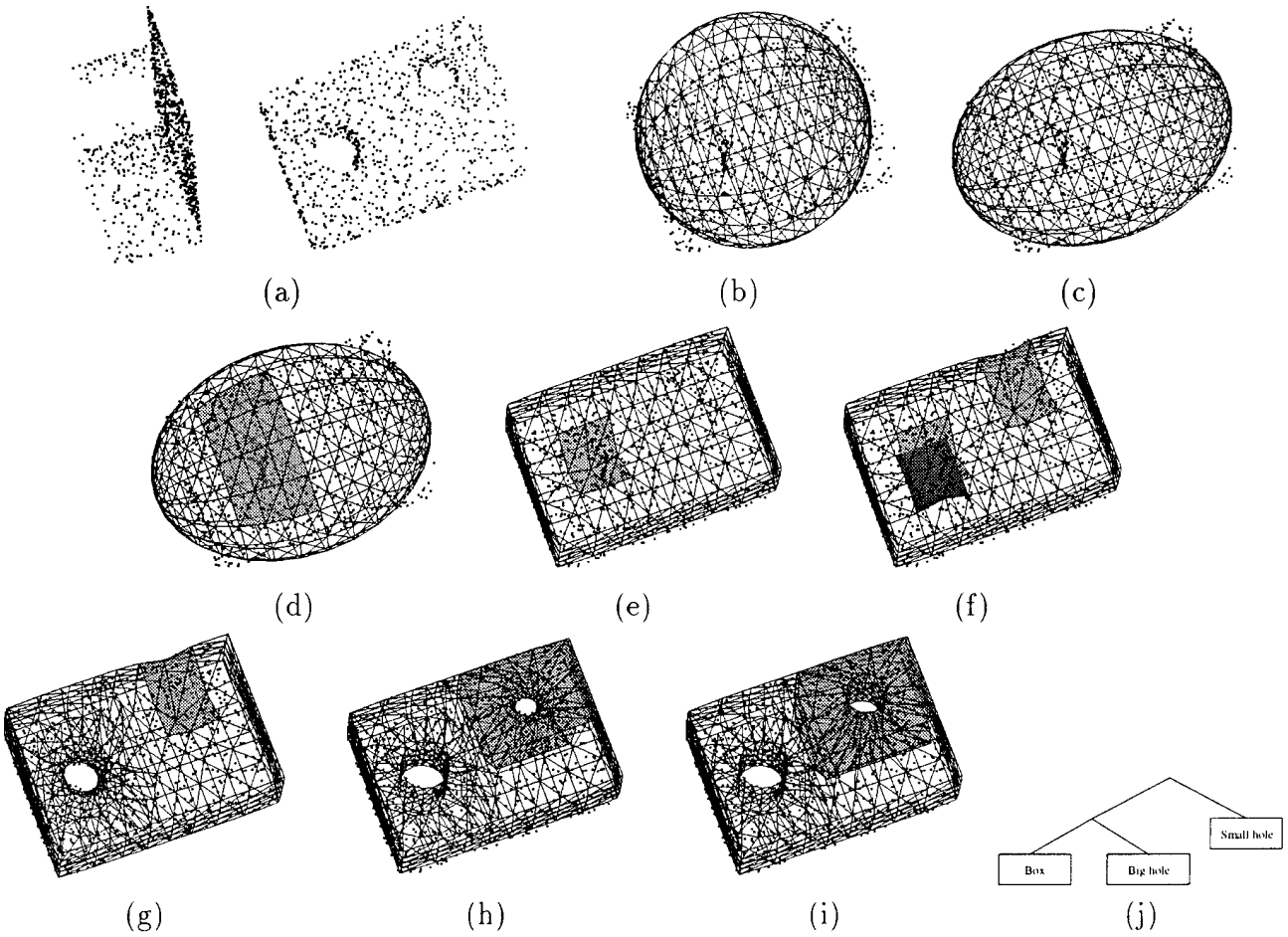


Figure 15: Fitting of a two holed object (a) range data (two views to show sparsity) (b) initial model (c) rough fit (d) blending region added (e) indentation formed (f) two more indentations formed (g) hole formed between two indentations (h) second hole formed (i) final shape (j) symbolic representation

References

[1] A. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

[2] I. Biederman. Recognition-by-components: a theory of human image understanding. *Psychological Review*, 94:115–147, April 1987.

[3] T. Binford. Visual perception by computer. In *IEEE Conference on Systems and Control*, December 1971.

[4] D. DeCarlo and D. Metaxas. Blended deformable models. In *Proceedings CVPR '94*, pages 566–572, 1994.

[5] H. Delingette. Simplex meshes: A general representation for 3d shape reconstruction. In *Proceedings CVPR '94*, pages 856–859, 1994.

[6] F. Ferrie, J. Lagarde, and P. Whaite. Darboux frames, snakes and superquadrics: Geometry from the bottom up. *IEEE Pattern Analysis and Machine Intelligence*, 15(8):771–784, 1993.

[7] A. Gupta and R. Bajcsy. Volumetric segmentation of range images of 3d objects using superquadric models. *Computer Vision, Graphics, and Image Processing*, 58:302–326, 1993.

[8] S. Han, D. Goldgof, and K. Bowyer. Using hyperquadrics for shape recovery from range data. In *IEEE Proceedings ICCV '93*, pages 492–496, June 1993.

[9] A. J. Hanson. Hyperquadrics: smoothly deformable shapes with convex polyhedral bounds. *Computer Vision, Graphics, and Image Processing*, 44:191–210, 1988.

[10] M. W. Hirsch. *Differentiable Topology.* Springer-Verlag, 1991.

[11] E. Koh, D. Metaxas, and N. Badler. Hierarchical shape representation using locally adaptive finite elements. In *Proceedings ECCV '94*, pages 441–446, May 1994.

[12] G. C. Lee and G. C. Stockman. Obtaining registered range and intensity images using the Technical Arts scanner. Technical Report CPS-91-08, Dept. of Computer Science, Michigan State University, 1991.

[13] A. Leonardis, F. Solina, and A. Macerl. A direct recovery of superquadric models in range images using recover-and-select paradigm. In *Proceedings ECCV '94*, pages 309–318, 1994.

[14] C. Liao and G. Medioni. Simulataneous segmentation and approximation of complex patterns. In *Proceedings CVPR '94*, pages 617–623, 1994.

[15] R. Malladi, J. A. Sethian, and B. C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Pattern Analysis and Machine Intelligence*, 1994, to appear.

[16] D. Manocha and J. F. Canny. A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, pages 491–516, 1991.

[17] D. Marr and K. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings Royal Society London*, 200:269–294, 1978.

[18] D. Metaxas. *Physics-Based Modeling of Nonrigid Objects for Vision and Graphics.* PhD thesis, Department of Computer Science, University of Toronto, 1992.

[19] D. Metaxas and D. Terzopoulos. Dynamic deformation of solid primitives with constraints. *IEEE Pattern Analysis and Machine Intelligence*, 15(6):569–579, June 1993.

[20] V. Nalwa. *A Guided Tour of Computer Vision.* Addison Wesley, 1993.

[21] A. Pentland. Perceptual organization and the representation of natural form. *Artificial Intelligence*, 28:293–331, 1986.

[22] A. Pentland and S. Sclaroff. Closed-form solutions for physically based shape modeling and recognition. *IEEE Pattern Analysis and Machine Intelligence*, 13(7):715–729, 1991.

[23] M. Rutishauser, M. Stricker, and M. Trobina. Merging range images of arbitrarily shaped objects. In *Proceedings CVPR '94*, pages 573–580, 1994.

[24] F. Solina and R. Bajcsy. Recovery of parametric models from range images: The case for superquadrics with global deformations. *IEEE Pattern Analysis and Machine Intelligence*, 12(2):131–147, 1990.

[25] R. Szeliski, D. Tonnesen, and D. Terzopoulos. Modeling surfaces of arbitrary topology with dynamic particles. In *Proceedings CVPR '93*, pages 82–87, 1993.

[26] D. Terzopoulos and D. Metaxas. Dynamic 3D models with local and global deformations: Deformable superquadrics. *IEEE Pattern Analysis and Machine Intelligence*, 13(7):703–714, 1991.

[27] D. Terzopoulos, A. Witkin, and M. Kass. Constraints on deformable models: Recovering 3D shape and nonrigid motion. *Artificial Intelligence*, 36(1):91–123, 1988.

[28] B. C. Vemuri and A. Radisavljevic. Multiresolution stochastic hybrid shape models with fractal priors. *ACM Transactions on Graphics*, 13(2):177–207, 1994.

C Integrating Anatomy and Physiology for Behavior Modeling: DeCarlo, Kaye, Metaxas, Clarke, Webber, and Badler

Integrating Anatomy and Physiology for Behavior Modeling*

Douglas DeCarlo, Jonathan Kaye, Dimitri Metaxas, John R. Clarke,
Bonnie Webber, Norm Badler
Center for Human Modeling and Simulation, University of Pennsylvania

Abstract. In producing realistic, animatable models of the human body, we see much to be gained from developing a *functional anatomy* that links the anatomical and physiological behavior of the body through fundamental causal principles. This paper describes our current Finite Element Method implementation of a simplified lung and chest cavity during normal quiet breathing and then disturbed by a simple pneumothorax. The lung model interacts with the model of the chest cavity through applied forces. The models are modular, and a second lung and more complex chest wall model can be added without disturbing the model of the other lung. During inhalation, a breathing force (corresponding to exertion of the diaphragm and chest wall muscles) is applied, causing the chest cavity to expand. When this force is removed (at the start of exhalation), the stretched lung recoils, applying pressure forces to the chest wall which cause the chest cavity to contract. To simulate a simple pneumothorax, the intrapleural pressure is set to atmospheric pressure, which removes pressure forces holding the lung close to the chest cavity and results in the lung returning to its unstretched shape.

1 Introduction

For some time now, we at the Center for Human Modeling and Simulation at the University of Pennsylvania have been developing human behavior models for virtual agents in simulated worlds. One underlying philosophy has been that producing realistic behavior involves both presenting our agents with accurate visual graphics and endowing them with structural and functional constraints of the body, as they interact with their world.

To produce analogous realism in our virtual agents' bodies, we recognize the critical relationship between the physical existence of an anatomical part with the functional role(s) it plays. This has led us to to couple our quantitative deformable model techniques [6] with models of the physiological mechanisms that produce physical changes. This allows us to design models that reflect the fact that anatomical parts have two intrinsic, interrelated existences: they are physical objects that obey physical laws, and they are part of physiological systems, so their behavior contributes to the overall functioning of the body. They are interrelated because changes to one can affect the other, both as part of the same system and as a result of physical adjacency.

*This work has been supported by the National Library of Medicine under grant number NO1–LM–4–3515.

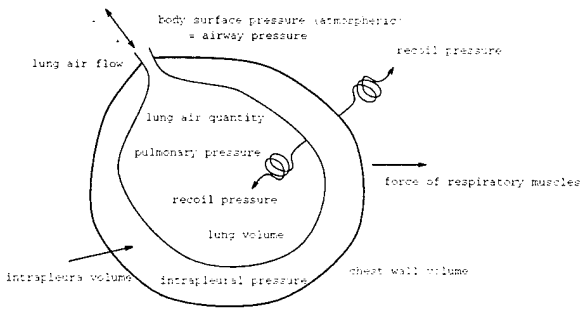


Figure 1: Qualitative, idealized lung model

1.1 The Effort

For several years we have been working to provide computer-based decision support in aid of the initial definitive management of multiple trauma [2, 9, 12]. Because trauma disrupts physiological processes through anatomy, we see much to be gained from creating a *functional anatomy* that will aid in visualizing and predicting the results of penetrating trauma to the human torso. A functional anatomy links the anatomical and physiological behavior of the body using fundamental, causal principles. Another area of medicine that can gain from models of functional anatomy is the emerging field of Virtual Surgery. To date, however, many efforts have concentrated on providing realistic images and dynamics (e.g., Noar [7] on techniques used in endoscopic simulators), neglecting functional issues. The consequence is that a necessary characteristic for virtual surgical simulators [10], *reactivity*, that organs must react appropriately to manipulation or cutting, such as by bleeding or leaking fluid, cannot be achieved.

We chose first to model the respiratory mechanism since it involves physiological change, such as pressures and flows, that depends on gross anatomical deformations. Ultimately, with models for other physiological systems [8], we want to demonstrate their interaction due to the physical space they share.

We currently express physiological dynamics in both a quantitative and qualitative framework. This provides us with a mixed quantitative/qualitative description of physiological behavior, which may be more appropriate for explanation than a purely quantitative model. Our system consists of two integrated levels of abstraction: (i) geometric and physics-based modeling of anatomy (shape extraction, motion, deformations, and graphical rendering); and (ii) simulation of physiological mechanisms that behave in accordance with physical laws and physiological processes.

This paper describes our ongoing effort [3] involving the procedures we have developed that graphically demonstrate organ geometry, physics, and physiological dynamics. It details our current implementation of a simplified lung during normal, quiet breathing, describing the quantitative results of our Finite Element Method implementation, based on techniques from [6].

Our qualitative formulation of the dynamics involved in quiet breathing is described in [3]. It makes use of the qualitative simulation paradigm QSIM [4]. Figure 1 shows the quantities we considered, assuming constant resistance and compliance.

2 Quantitative Lung Modeling

Our anatomical modeling is based on our physics–based framework [6, 11] for shape and nonrigid motion estimation and synthesis. This framework features a Lagrangian dynamics framework which will be used to describe the dynamics of our lung model. The geometry of the lung model will be chosen so that we may utilize these previously developed methods for deformable body mechanics.

When applying Lagrangian dynamics [6], we obtain second order equations of motion which take the general form

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{g}_\mathbf{q} + \mathbf{f}_\mathbf{q}, \tag{1}$$

where $\mathbf{q}$ are the generalized coordinates (the degrees of freedom) of the model, $\mathbf{M}$, $\mathbf{D}$, and $\mathbf{K}$ are the mass, damping, and stiffness matrices, respectively, $\mathbf{g}_\mathbf{q}$ are inertial forces, and $\mathbf{f}_\mathbf{q}$ are the generalized external forces. The vector $\mathbf{q}$ contains the information needed to specify the shape of the lung. If the equation of motion (1) is integrated over time, the dynamic deformation of the model is observed.

The shape and viscoelastic properties of the lungs are modeled by using isoparametric finite elements [13]. Because we currently lack detailed data on the elastic properties of the viscoelastic material of the lung, we use linear finite elements. However, our methodology is general and is also applicable to non-linear finite elements.

We incorporate the isoparametric finite elements into the Lagrange equations of motion (1). The finite elements are used to compute internal elastic forces that arise due to the deformation of the lung. This deformation is caused by applied forces that include pressure and collision forces.

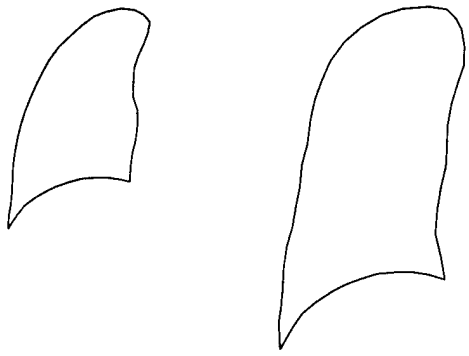
We can now use this model to create dynamic simulations of inhalation and exhalation, and also of a simple pneumothorax. Our model of the lung interacts with the surrounding chest cavity by applied forces. This type of representation of the lung lends itself to a modular approach, where this lung model could be incorporated into a larger model of human anatomy without changing the implementation. For instance, the addition of a second lung or heart would not require changes in the implementation of the first lung. The first lung need not "know" about the other lung or the heart directly. Their effects could be dealt with through applied forces.

2.1 Model Geometry

The following sections describe the lung model used in the dynamic simulation. While the model is two-dimensional, it observes the qualitative behaviors of a full lung model in three dimensions.

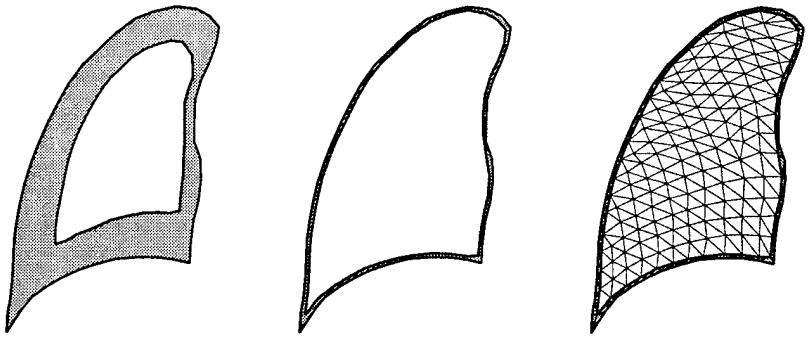
This lung model will react to external forces that include pressure forces (due to the difference in pressure between the lung and intrapleural space), and contact forces caused by the lung rubbing against the chest wall.

The lung is contained within the chest cavity. For this implementation, the chest cavity is given two degrees of freedom, which correspond to diaphragm and chest wall deformation. This simplified chest model is suitable currently, for the purpose of demonstration. However, because of the modularity of our lung model, we could add a more complex chest wall model without disturbing our model of the lung. This chest cavity model contains what is necessary for simulation of inhalation and exhalation. Figure 2 shows kinematic motion of the chest wall at the extremes–(a) at rest, and (b) fully inhaled.



(a) (b)

Figure 2: Geometry of the chest wall (a) at rest, and (b) after inhalation



(a) (b) (c)

Figure 3: Dynamic construction of lung model (a)-(b), and finite element mesh geometry (c)

Within this chest wall is the finite element mesh for the lung and intrapleural space. The initial model can be constructed by creating the lung in its unstretched shape shown in figure 3(a), and then setting the intrapleural pressure to its negative value (with respect to atmospheric pressure), resulting in the lung at rest in (b) (recall that the lung is stretched in its normal resting state due to the negative intrapleural pressure). In this (and all following) model diagrams, the lung is shaded with a light gray, and the intrapleural space with a darker gray. A diagram of the finite element mesh used is shown in figure 3(c).

2.2 *Model Dynamics*

During inhalation, the increase in size of the lung is due to pressure forces. As the chest cavity increases in volume, the difference in pressure between the lung and intrapleural space causes the lung to expand. For the applications in this paper, we make the simplifying assumption that the pressure changes occur instantaneously. This means the lung is always at atmospheric pressure, and the intrapleural space has one common pressure, P, which changes assuming PV is constant (where V is the intrapleural volume).

The forces that arise due to pressure differences occur at the boundaries of the intrapleural space, which is the only location where two adjacent elements will differ in pressure. For such an element, we can compute the pressure force at a node along an element edge as

$$f_{\text{pressure}} = \frac{P}{l}\mathbf{n} \tag{2}$$

where P is the pressure of an element, l is the length of the edge, and $\mathbf{n}$ is a normal pointing

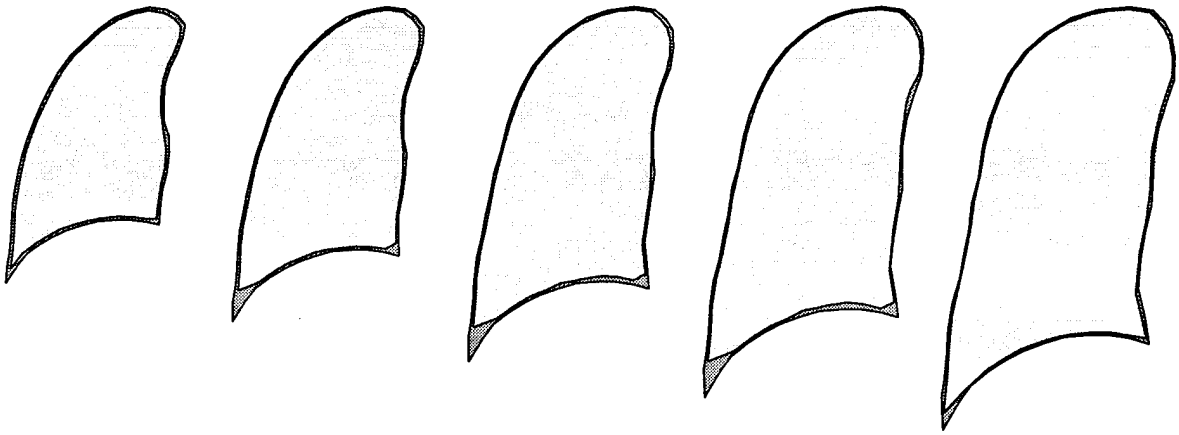


Figure 4: Inhalation of the lung

out of the element that is perpendicular to the edge. This is a 2-D analog of the $f = P/\text{area}$ relationship. When considering the sum of all pressure forces, the net contribution of force for each edge is related to the pressure gradient across the edge.

2.2.1 Chest Cavity Dynamics

The deformation of the chest cavity is performed by specifying its two degrees of freedom– diaphragm and chest wall shape. The pressure forces described above not only are applied to the lung, but also act on the chest wall. Using our Lagrangian dynamics framework, these applied forces are converted into generalized forces which directly control the shape of the chest cavity. During inhalation, a breathing force (corresponding to exertion of the diaphragm and chest wall muscles) is applied, causing the chest cavity to increase in size. Once this force is removed (at the start of exhalation), the stretched lung recoils, applying pressure forces to the chest wall, causing the chest cavity to decrease in volume.

The contact forces due to collision of the lung with the chest wall are implemented as in [5].

2.3 Simulations

The following are simulations using the dynamic lung model described above. These simulations run at interactive rates on a 100 MHz R4000 VGX SGI. Figure 4 shows a simulation showing the process of inhalation using our model. The behavior of this model agrees qualitatively with our qualitative physiological model.

To simulate a simple pneumothorax, we can set the intrapleural pressure to be equivalent to body surface pressure (atmospheric pressure) in our model. This will eliminate any pressure forces holding the lung close to the chest cavity. The resulting collapsing motion of the lung is shown in figure 5. The location of the injury is indicated by the gap in the chest wall. Notice how the lung returns to its unstretched shape, as shown in figure 2(a).

3 Conclusion

Our work involves the simulation, modeling, and visualization of anatomical and physiological mechanisms, considering in particular pathology related to penetrating injuries. Our intent is to provide a reusable anatomical knowledge base coupled directly with knowledge of the

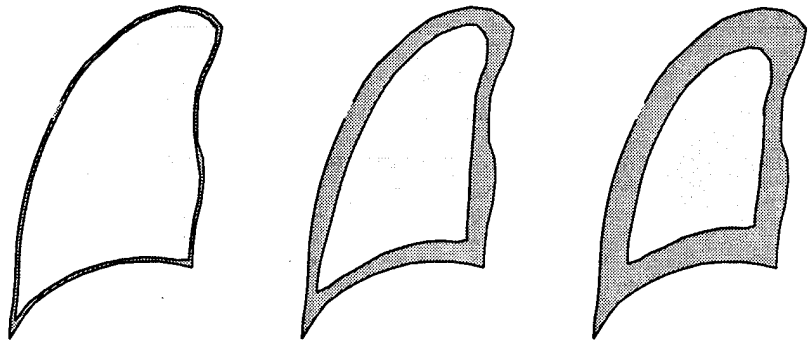


Figure 5: Simple pneumothorax of the lung

underlying physiology, or what we refer to as a *functional anatomy*. A functional anatomy links the anatomical and physiological behavior of the body through fundamental causal principles.

In this paper, we have examined one aspect of our project, the Finite Element Method implementation of a two-dimensional, idealized lung. This preliminary work will be used as the basis for our continuing physics-based three-dimensional modeling which we will apply to simulating normal respiratory physiology and related pathologies. It will also serve as a basis for considering the interaction of physiological systems due volume constraints.

References

[1] N. Badler, C. Phillips, and B. Webber. *Simulating Humans: Computer Graphics, Animation, and Control*. New York: Oxford University Press, 1993.

[2] Clarke, J.R., Webber, B., Niv, M., Rymon, R., Gertner, A. and Kaye, J. The Care of Injured Patients: An architecture of Medical Knowledge. *Der Chirurg* (Special issue on surgical decision-making), April 1994.

[3] J. Kaye, D. Metaxas, J. R. Clarke, B. Webber. Lung Modeling: Integrating Anatomy and Physiology. *First International Symposium on Medical Robotics and Computer–Assisted Surgery (MRCAS-94)*, Pittsburgh, 1994.

[4] B. Kuipers. Qualitative Simulation. *Artificial Intelligence*, **29**: pp. 289–338, 1986.

[5] D. Metaxas. *Physics-Based Modeling of Nonrigid Objects for Vision and Graphics*. Ph.D. thesis, Department of Computer Science, University of Toronto, 1992.

[6] D. Metaxas and D. Terzopoulos. Shape and Nonrigid Motion Estimation Through Physics–Based Synthesis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, **15**(6): pp. 569–579, June 1993.

[7] M.D. Noar. Endoscopy Simulation: A Brave New World? *Endoscopy*, **23**: pp. 147-9, 1991.

[8] G. Provan, S. Neumann, J. R. Clarke, and J. Kaye. A Mixed Qualitative/Quantitative Acute Cardiovascular Model: Preliminary Report, for the *AAAI Spring Symposium, AI in Medicine*, Stanford, 1994.

[9] Rymon, R., Webber, B. L. and Clarke, J. R., Progressive Horizon Planning – Planning Exploratory-Corrective Behavior. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6). Special issue on Planning, Scheduling and Control, November 1993.

[10] R. M. Satava. Virtual reality surgical simulation: the first steps. *Surgical Endoscopy*, **7**: pp. 203–205, 1993.

[11] D. Terzopoulos and D. Metaxas. Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics. *IEEE Trans. Pattern Analysis and Machine Intelligence*, **13**(7): pp. 703–714, 1991.

[12] Webber, B., Rymon, R. and Clarke, J.R. Flexible Support for Trauma Management through Goal-directed Reasoning and Planning. *Artificial Intelligence in Medicine* **4**(2): pp. 145-163, April 1992.

[13] O. Zienkiewicz. *The Finite Element Method.* McGraw-Hill, 1977.

D Volumetric Deformable Models with Parameter Functions: A New Approach to the 3D Motion Analysis of the LV from MRI-SPAMM: Park, Metaxas, and Axel

Volumetric Deformable Models with Parameter Functions: A New Approach to the 3D Motion Analysis of the LV from MRI-SPAMM

Jinah Park[1], Dimitri Metaxas[1] and Leon Axel[2]

[1] Dept. of Computer & Information Science, U. of Pennsylvania, Philadelphia, PA 19104
[2]Dept. of Radiology, U. of Pennsylvania, Philadelphia, PA 19104
e-mail: jinah@gradient.cis.upenn.edu, dnm@central.cis.upenn.edu

Abstract

We present a new method for analyzing the 3D motion of the heart's left ventricle (LV) from tagged magnetic resonance imaging (MRI) data. Our technique is based on the development of a new class of volumetric physics-based deformable models whose parameters are functions and allow the definition of new parameterized primitives and parameterized deformations which can capture the local shape variation of a complex volumetric object. These parameters require no complex post-processing in order to be used by a physician. Using a physics-based approach, we convert these volumetric geometric models into deformable models that deform due to forces exerted from the data points. We present a new technique for calculating forces exerted by tagged MRI data to material points of the deformable model. These new volumetric models allow the accurate estimation of the shape and motion of the inner and outer walls of the LV, and the shape and motion within the walls. We present experiments involving the extraction of shape and motion of the LV during systole. Furthermore, by plotting the variations over time of the extracted LV model parameters from normal heart data we are able to quantitatively analyze and compare the epicardial and endocardial motion.

Key words: Medical Computer Vision, Motion Analysis, Shape Representation, Volumetric Deformable Models with Parameter Functions, Physics-Based Modeling.

1 Introduction

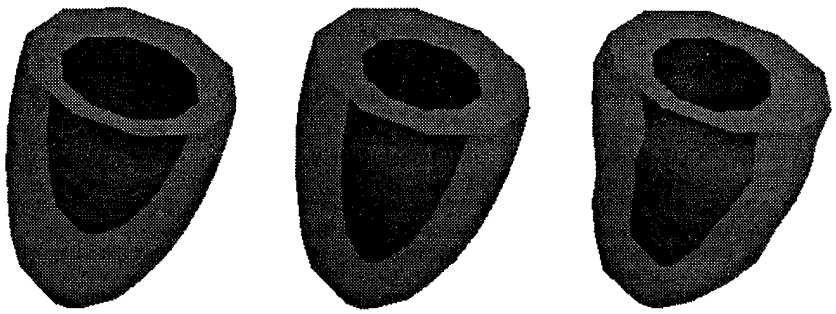
Characterization of heart wall motion on a regional level is required to understand cardiac mechanics and the processes of underlying disease such as ischemia. Previously, material points on the myocardium have been located and tracked in order to accurately measure heart wall motion. Such marker-based methods have included the implantation of beads [21] or ultrasonic crystals [20], use of naturally occurring landmarks [22], and MR tagging [26]. SPAMM (SPAtial Modulation of Magnetization) [1], which is a magnetic resonance imaging technique with magnetic tagging, has been developed at the University of Pennsylvania. This technique has been used to demonstrate regional motion patterns during systole[16, 2], and the methods for calculating material deformation have been validated using deformable phantoms [24]. The advantage of the SPAMM technique is that a number of material points can be marked in a very short time with a simple procedure, and can be tracked during systole in a non-invasive setting, providing temporal correspondence of material points. This correspondence in conjunction with the use of the three dimensional location of each tagged point can be subsequently used as input to a motion analysis technique to extract the three dimensional motion parameters.

Recently, computer vision techniques for reconstructing the 3D *surface* shape and motion (endocardial) of the left ventricle (LV) of the heart from CT or MRI data have been developed [9, 6, 18, 3, 17, 4, 13] and are based on the use of finite elements, spring-mass systems, deformation modes, bending and stretching thin-plate models, and other physics-based or geometric techniques. One limitation of the above techniques is that they do not capture the twisting motion of the heart, known to occur during systole. Also, they are formulated in terms of either many local parameters that require non-trivial processing to be useful to a physician, or very few parameters that can offer only a gross approximation of the motion of a heart. To overcome the problems of the above techniques in terms of accurately estimating the LV surface shape and motion and in order to extract parameters that can be easily interpreted by physicians Park *et al.* [14] developed a new class of surface deformable primitives whose global parameters are functions. These new models can capture the axial twisting, bending, and contraction of the LV surface. The input to these models were datapoints sampled from the mid-wall of the 3D finite element model of Young and Axel [23], which estimated the LV shape and motion from MRI-SPAMM.

However, the LV motion can't be captured entirely with surface models because the endocardial and epicardial motions are sufficiently different. Recently, techniques for analyzing the *volumetric* motion of the LV have also been developed. Young and Axel [23] and Moore *et al.* [12] used 3D finite elements and SPAMM data, while Denney and Prince [5] used a multidimensional stochastic model and tagged MR image sequences. The main limitation of these techniques is that there is an enormous amount of information on motion and deformation captured. In [23] the three-dimensional strain tensor, for example, has three normal components and three shear components, each of which may vary with position in the wall as well as over time. In order to understand the complex relationship between these components and other motion parameters, it is desirable to characterize the motion in terms of a few physical parameters that offer sufficient accuracy.

In this paper we present a new general class of volumetric primitives for estimating and ana-

lyzing the full motion of the LV from MRI-SPAMM data. Through our technique we estimate the deformation and motion of the LV in terms of a few "global" *parameter functions*, such as twisting, whose value is allowed to vary locally. In this way the complex motion of the heart is described by the same small number of parameters, which vary from region to region. Therefore, we can capture the shape and motion 1) of the LV walls and, 2) in between the walls. Furthermore, these parameters are intuitive and can be used by a physician without further complex processing. These new volumetric deformable primitives are parameterized using global parameter functions whose value varies across the shape of the primitives as opposed to being constant [10, 19]. Through the use of appropriate parameterization the axes of the deformable primitives can be curved. This is a generalization compared to other parameterized volumetric primitives used in computer vision. Fig. 1(c) shows an example of the new family of volumetric deformable primitives with parameter functions, used in estimating the volumetric shape of the LV.



(a) (b) (c)

Figure 1: Volumetric Deformable Models

While these new shape primitives can be used in many applications, we here present shape and motion estimation results for the LV. By incorporating the geometric definition of the models into our physics-based framework [10], we create dynamic models that deform due to forces exerted from 3D MRI-SPAMM datapoints and conform to the given dataset. The initial shape of the LV is captured based on contour information from the inner and outer walls of the LV, which is extracted using snakes [7]. Given that the geometry of the MRI-SPAMM data is such that the given data lie in sets of orthogonal planes, we develop a new algorithm for extracting forces from the given data to material points within our volumetric model. By solving a cubic equation, we first define the model's material coordinate where the computed force is being exerted. Then we extrapolate the computed force to the nodes of the corresponding volumetric finite element. Subsequently, we use the physics-based methodology developed in [10] to estimate the values of the parameter functions of our model. The LV extracted parameters can then be directly used for analysis by a physician after plotting parameter graphs.

We applied our technique to normal subjects and analyzed the results of our parameter extraction. These results quantitatively verified qualitative knowledge about the LV known to physicians. Furthermore, we present a method for visualizing the model fitting results.

2 Volumetric Deformable Models with Parameter Functions

2.1 Geometry

Our new class of deformable models allows the use of global parameters that can characterize a volumetric shape in terms of a few parameter functions.

The material coordinates of our model, $\mathrm{u} = (u, v, w)$, are defined in a three-dimensional domain Ω, and the positions of points on the model relative to an inertial frame of reference Φ in 3D space are given by a vector-valued, time-varying function $\mathbf{x}(\mathrm{u}, t) = (x(\mathrm{u}, t), y(\mathrm{u}, t), z(\mathrm{u}, t))^\mathsf{T}$, where $^\mathsf{T}$ denotes transposition. We set up a non-inertial, model-centered reference frame ϕ and express the position of a point on a model as

$$\mathbf{x} = \mathbf{c} + \mathbf{R}\mathbf{s},$$

where the center of the model $\mathbf{c}(t)$ is the origin of ϕ and the rotation matrix $\mathbf{R}(t)$ gives the orientation of ϕ relative to Φ with a reference shape $\mathbf{s}$. Thus, $\mathbf{s}(\mathrm{u}, t)$ gives the positions of points on the model relative to the model frame. Local deformations [10] are not used, since the global deformations $\mathbf{s}$ will be defined based on parameter functions capable of capturing the local variation of the LV shape.

We define the reference shape as

$$\mathbf{s} = \mathbf{T}(\mathbf{e}; \beta_0(\mathrm{u}), \beta_1(\mathrm{u}), \ldots),$$

where $\mathbf{e}$ can represent either a set of 3D points in space[1] or a geometric primitive $\mathbf{e}(\mathrm{u}; \alpha_0(\mathrm{u}), \alpha_1(\mathrm{u}), \ldots)$ defined parametrically in u and parameterized by the variables $\alpha_i(\mathrm{u})$. The shape represented by $\mathbf{e}$ is subjected to the *deformation* $\mathbf{T}$ which depends on the deformation parameter functions $\beta_i(\mathrm{u})$. Although generally nonlinear, $\mathbf{e}$ and $\mathbf{T}$ are assumed to be differentiable[2] so that we may compute the Jacobian of $\mathbf{s}$. $\mathbf{T}$ may be a composite sequence of primitive deformation functions $\mathbf{T}(\mathbf{e}) = \mathbf{T}_1(\mathbf{T}_2(\ldots\mathbf{T}_n(\mathbf{e})))$. We concatenate the deformation parameters into the vector $\mathbf{q}_s$:

$$\mathbf{q}_s = (\alpha_0(\mathrm{u}), \alpha_1(\mathrm{u}), \ldots, \beta_0(\mathrm{u}), \beta_1(\mathrm{u}), \ldots)^\mathsf{T}.$$

The parameters α_i and β_i are functions of u, instead of constants. This definition allows us to generalize definitions of volumetric primitives (e.g., volumetric superquadrics, cubes) and parameterized deformations (e.g., twisting), as will be shown in the following section and was demonstrated for surface models in [14]. Our technique for creating volumetric primitives with parameter functions can be applied to any parametric primitive, by replacing its constant parameters with differentiable parameter functions.

For the applications in this paper, the orientation of the deformable model is schematically drawn in Fig. 2. The model-centered reference frame ϕ is chosen at the center of the LV with the y-axis pointing towards the right ventricle (RV). The material coordinates $\mathrm{u} = (u, v, w)$ are depicted in Fig. 2(b), where u runs from the apex to the base of the LV, v starts and ends at the point

[1] In that case, the material coordinates u coincide with the Cartesian space in which the 3D points are expressed.
[2] In the case where $\mathbf{e}$ is a set of points, the above assumption does not apply.

where the septum is located, and w is used for the definition of model points between the inner and outer walls of the deformable model. We will also assume that $\alpha_i(\mathrm{u}) = \alpha_i(u, w)$, $\beta_i(\mathrm{u}) = \beta_i(u, w)$. Fig. 1(a) is an example of the volumetric model with constant parameters, and Figs. 1(b-c) are examples of two volumetric models whose parameters are functions of w and of (u, w), respectively.

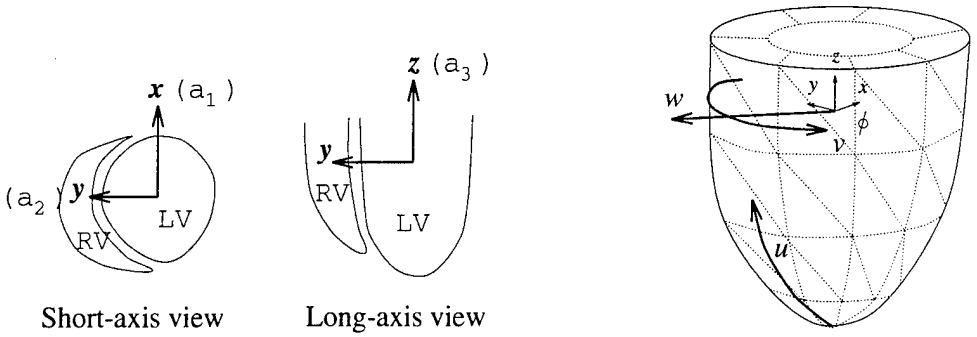


Figure 2: Orientation of a model

To create a volumetric model for the LV, we first define a generalized volumetric primitive $\mathbf{e} = (e_1, e_2, e_3)^{\mathsf{T}}$ as follows[3]:

$$\mathbf{e} = \mathbf{e}(\mathrm{u}; a_1(\mathrm{u}), a_2(\mathrm{u}), a_3(\mathrm{u})) \;=\; a_0 w \begin{pmatrix} a_1(\mathrm{u}) \cos u \cos v \\ a_2(\mathrm{u}) \cos u \sin v \\ a_3(\mathrm{u}) \sin u \end{pmatrix}, \tag{1}$$

where $-\pi/2 \le u \le \pi/4$, $-\pi \le v < \pi$, $a_0(u) > 0$, and $0 \le a_1(u), a_2(u), a_3(u) \le 1$. a_0 is a scale parameter and a_1, a_2 and a_3 are the aspect ratio parameters along the x-, y- and z-axis, respectively. Note that the ranges of the u and v parameters for our generalized volumetric primitive are defined (see (1)) in order to construct an open volumetric parameterized primitive. To define our model we further add parameterized twisting and axis offset deformations.

Given the above defined primitive $\mathbf{e} = (e_1, e_2, e_3)$, we define the parameterized twisting along the model axis z, which results in the global shape $\mathbf{s}_t = (s_1, s_2, s_3)^{\mathsf{T}}$:

$$\mathbf{s}_t = \mathbf{T}_t(\mathbf{e}; \tau(u, w)) \;=\; \begin{pmatrix} e_1 \cos(\tau(u, w)) - e_2 \sin(\tau(u, w)) \\ e_1 \sin(\tau(u, w)) + e_2 \cos(\tau(u, w)) \\ e_3 \end{pmatrix}, \tag{2}$$

where $\tau(u, w)$ is the twisting parameter function along the axis z. Finally, we apply offset deformations which allow the axis to be non-straight in the x and y directions. In this way we can recover

[3]This is a generalization of the definition of an ellipsoid.

the LV shape more accurately. The resulting reference shape $\mathbf{s}$ is expressed as follows:

$$\mathbf{s} = \mathbf{T}_o(\mathbf{T}_t(\mathbf{e}; e_{1_o}(u,w), e_{2_o}(u,w)) = \begin{pmatrix} s_1 + e_{1_o}(u,w) \\ s_2 + e_{2_o}(u,w) \\ s_3 \end{pmatrix}, \quad (3)$$

where $e_{1_o}(u,w)$ and $e_{2_o}(u,w)$ are axis-offset parameter functions in the x and y directions, respectively. The model parameter functions $\mathbf{q}_s$ used in this paper are piecewise linear functions, to avoid smoothing C^0 regions of the LV.

2.2 Kinematics and Dynamics of the System

The velocity of points on the model is given by [10]:

$$\dot{\mathbf{x}} = [\mathbf{I} \ \ \mathbf{B} \ \ \mathbf{RJ}]\dot{\mathbf{q}} = \mathbf{L}\dot{\mathbf{q}}, \quad (4)$$

where $\mathbf{L}$ is the model Jacobian matrix which maps the model's parameter space into the 3D space, $\mathbf{q} = (\mathbf{q}_c^\top, \mathbf{q}_\theta^\top, \mathbf{q}_s^\top)^\top$ is the vector of the model's degrees of freedom, $\mathbf{q}_c = \mathbf{c}$ and $\mathbf{q}_\theta$ is the vector of the rotational degrees of freedom expressed as a quaternion. Finally, $\mathbf{B} = [...\partial(\mathbf{Rp})/\partial\theta_i...]$ and $\mathbf{J}$ is the Jacobian of the reference shape $\mathbf{s}$.

We can make our model dynamic in $\mathbf{q}$ by introducing a deformation strain energy [10]. The mass density was set to zero for the shape recovery application so that the system has no inertia. The resulting simplified dynamic equations of motion are

$$\mathbf{D}\dot{\mathbf{q}} = \mathbf{f}_q, \quad (5)$$

where $\mathbf{D}$ is the damping matrix and $\mathbf{f}_q(\mathrm{u}, t)$ are the generalized external forces associated with the degrees of freedom of the model. This equation yields a model that comes to rest when all the applied forces equilibrate or vanish. We also decouple the equations by assuming that $\mathbf{D}$ is diagonal and constant over time. We employ an adaptive-step first-order Euler method to integrate (5). Given that the MRI-SPAMM data are relatively accurate and to avoid undesired smoothing caused by the model, we did not introduce any internal stiffness, $\mathbf{K}$, to the global parameters of our model.

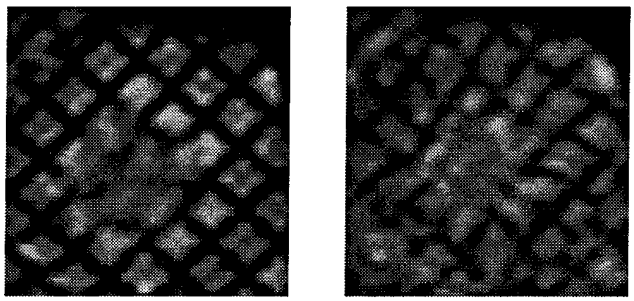
The generalized forces $\mathbf{f}_q$ are computed using the formula

$$\mathbf{f}_q = \int \mathbf{L}^\top \mathbf{f} \, d\mathrm{u}.$$

These forces are associated with the components of $\mathbf{q}$, where $\mathbf{f}(\mathrm{u}, t)$ is the 3D force distribution applied to the model.

3 Boundary and SPAMM Data

The data were obtained from the Department of Radiology, University of Pennsylvania and were collected during the LV systole over 5 intervals. The SPAMM data collection technique is based on the application prior to imaging of a saturation pulse sequence where the amplitude of the magnetization varies spatially, in a sinusoidal-like fashion. This saturation pulse sequence forms the *tagging planes*. At the minima of this sinusoidal-like variation of the magnetization, dark lines appear in the *image plane* which intersects the tagging planes. If we continue to image the tissue after the saturation pulse sequence is applied, we can see those dark lines move, allowing us to track the motion of the underlying tissue. To track points instead of lines, we apply another set of saturation pulse sequences which form a set of tagging planes orthogonal to the previous set of tagging planes. The intersection on an image plane of the associated dark lines defines the SPAMM datapoints on this plane. Figs. 3(a) and (b) show short axis views of a LV at end-diastole and towards end-systole, respectively, where the SPAMM datapoints are defined by the intersections of the respective dark lines. The method for extracting these intersection points was described in [25].



(a) time=1 (b) time=4

Figure 3: SPAMM images

Given that every image plane is spatially fixed, while the LV moves, the through-plane motion of the SPAMM datapoints on every image plane cannot be captured. Fig. 4 shows the location of a SPAMM datapoint $\mathbf{S}$ at two different times t_1 and t_2. Initially, $\mathbf{S}(t_1)$ coincides with a material point $\mathbf{M}(t_1)$. However, the motion of the SPAMM datapoint between these two time-instances corresponds to the component on the image plane of the motion of the material point $\mathbf{M}$, which at time t_2 lies somewhere along the line of intersection of the tagging planes at time t_2 as shown in Fig. 4. Therefore, we need to combine two sets of mutually orthogonal image planes (e.g., short and long axis views) to estimate the 3D motion of the LV. These sets of SPAMM datapoints are shown in Fig. 8.

It is also important to mention that the SPAMM data in the two orthogonal sets of image planes do not correspond to the same material points, but to different material points. These observations will be used in the calculation of the forces exerted from the SPAMM data to the volumetric deformable model.

To determine the initial shape of the inner and outer walls of the LV, we use snakes [7] to

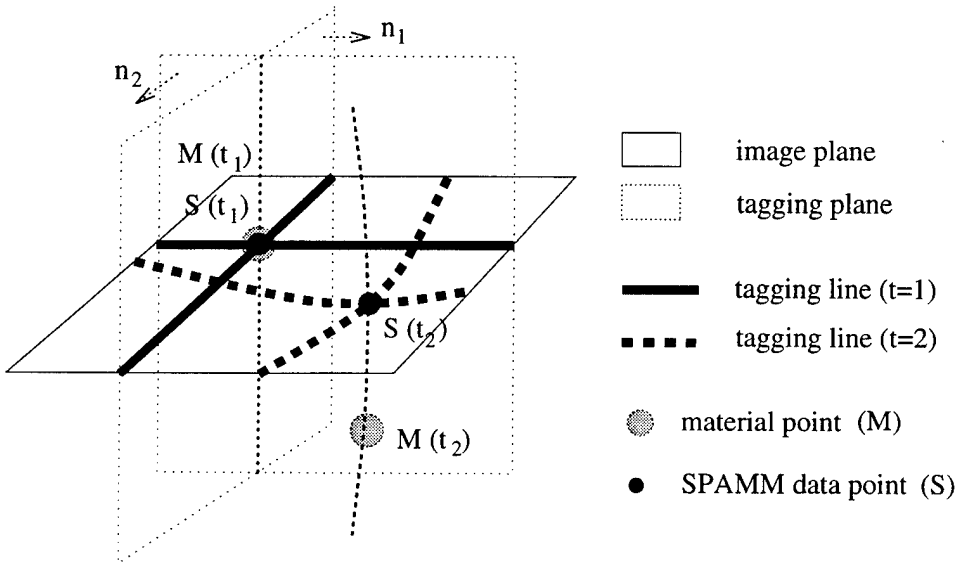


Figure 4: Tagging Planes and Image Planes

capture sets of 2D boundary contours from the two orthogonal sets of image planes. The boundary datapoints sampled from the contours at the initial time $t = 1$ (end-diastole) is shown in Fig. 9(a), where black dots are for the outer wall and white dots are for the inner wall.

The boundary and the SPAMM datapoints will exert forces on our model in order to estimate the shape and motion of the LV.

4 Model Force Computation from the Data

Depending on the type of data we compute the corresponding forces on the model in two different ways. We assume a triangular tessellation of the inner and outer walls of the volumetric primitives (as depicted in Fig. 2(b)) which are the faces of a prismatic tessellation of the volumetric model.

4.1 Force Computation from Boundary Data

Boundary data simply constrain the shape of the inner and outer walls of the LV and provide no correspondence of points over time. Therefore, we compute the forces from each boundary datapoint $\mathbf{P}$ to the corresponding model wall (inner or outer) as shown in Fig. 5. Approximating each boundary triangular element with a plane, we determine the element whose distance from $\mathbf{P}$ is minimum and we compute the intersection point $\mathbf{Q}$. The force that $\mathbf{P}$ exerts on the model is

$$\mathbf{f}_P = \gamma_1(\mathbf{P} - \mathbf{Q}), \tag{6}$$

where γ_1 is the strength of the force. We then distribute $\mathbf{f}_P$ to the nodes $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$ of the associated boundary triangular element based on the formula

$$\mathbf{f}_{\mathbf{x}_i} = m_i \, \mathbf{f}_P, \quad i = 1, 2, 3, \tag{7}$$

where the m_i are computed from the solution of the following linear system

$$\sum_i m_i \mathbf{x}_i = \mathbf{Q}. \tag{8}$$

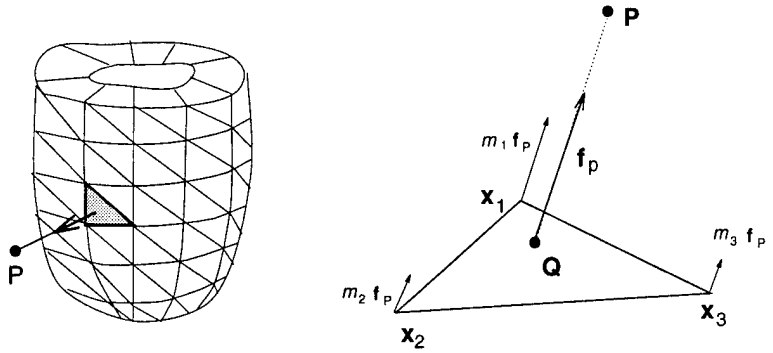


Figure 5: Forces from Boundary Datapoints.

4.2 Force Computation from SPAMM Data

As opposed to the boundary data, SPAMM data provide correspondence over time of the associated SPAMM points. Initially we assume that the SPAMM datapoints and the model material points coincide. Let $\mathbf{M}(t_1)$ be the material point which initially coincides with a SPAMM datapoint $\mathbf{S}(t_1)$ at time t_1 (see Fig. 6). Let also $\mathbf{S}(t_2)$ and $\mathbf{S}(t_3)$ be the corresponding SPAMM datapoints to the point $\mathbf{S}(t_1)$ at the next two time frames. Then the force on $\mathbf{M}(t_1)$ from $\mathbf{S}(t_2)$ is computed as

$$\mathbf{f}_{S(t_2)} = \gamma_2 \, [([\mathbf{S}(t_2) - \mathbf{M}(t_1)] \cdot \mathbf{n}_1) \, \mathbf{n}_1 + ([\mathbf{S}(t_2) - \mathbf{M}(t_1)] \cdot \mathbf{n}_2) \, \mathbf{n}_2], \tag{9}$$

where γ_2 is the strength of the force and $\mathbf{n}_1, \mathbf{n}_2$ are the unit normals of the corresponding initial (i.e., at time t_1) tagging planes as shown in Fig. 4.

That force will cause $\mathbf{M}(t_1)$ to move to a new position $\mathbf{M}(t_2)$. Subsequently, the force $\mathbf{f}_s(t_3)$ on $\mathbf{M}(t_2)$ from $\mathbf{S}(t_3)$ will be computed in a similar fashion and it is shown in Fig. 6.

Once we compute these force we distribute them to the nodes of the deformable model. These nodal forces will cause the model to deform. To distribute at any time frame t_i, the computed force $\mathbf{f}_S$ to the nodes of the prism $A_{out}B_{out}C_{out}A_{in}B_{in}C_{in}$ within which $\mathbf{M}$ lies, we do the following (see Fig. 7). Based on the finite element theory we want to compute a triangle ABC in which $\mathbf{M}$ lies such that

$$r = \frac{\mathbf{A} - \mathbf{A}_{in}}{\mathbf{A}_{out} - \mathbf{A}_{in}} = \frac{\mathbf{B} - \mathbf{B}_{in}}{\mathbf{B}_{out} - \mathbf{B}_{in}} = \frac{\mathbf{C} - \mathbf{C}_{in}}{\mathbf{C}_{out} - \mathbf{C}_{in}}, \tag{10}$$

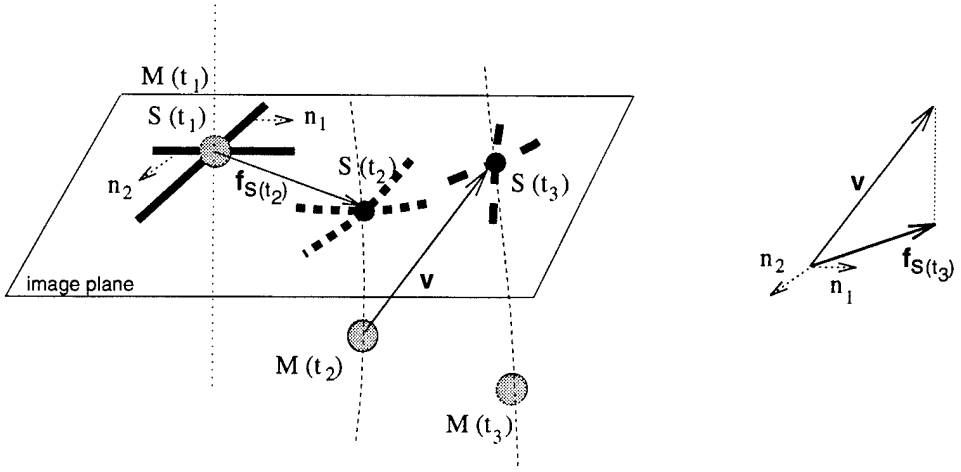


Figure 6: Forces $\mathbf{f}_s(t_2)$, $\mathbf{f}_s(t_3)$ from SPAMM points $\mathbf{S}(t_2)$ and $\mathbf{S}(t_3)$, respectively.

where r is a scalar. To compute r we solve the following cubic scalar equation using the Newton-Raphson method

$$(\mathbf{M} - \mathbf{A}) \cdot ((\mathbf{C} - \mathbf{A}) \times (\mathbf{B} - \mathbf{A})) = 0, \tag{11}$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are computed wrt to r from (10).

We extrapolate $\mathbf{f}_S$ to the nodes of triangle ABC in the same way as it was described in the force computation for the boundary data and we compute the scalars m_A, m_B, m_C which correspond to nodes $\mathbf{A}, \mathbf{B}, \mathbf{C}$, respectively. Then the forces to the nodes of the prism are computed as follows:

$$\begin{aligned} \mathbf{f}_{N_{out}} &= r\ m_N\ \mathbf{f}_S \\ \mathbf{f}_{N_{in}} &= (1-r)\ m_N\ \mathbf{f}_S, \end{aligned} \tag{12}$$

where $N = \{A, B, C\}$.

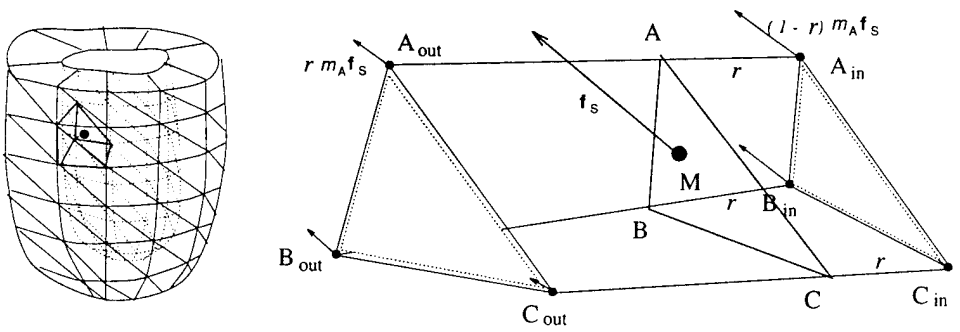
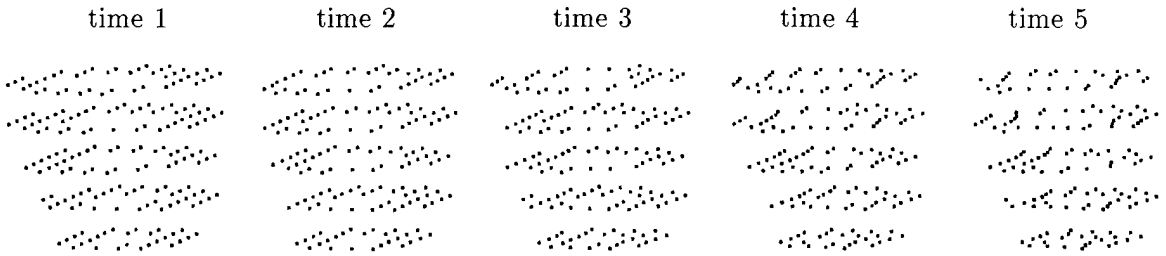


Figure 7: Distributing a Force from a SPAMM datapoint.

The computation of r which determines the model material point which corresponds to a SPAMM data is only done once at the beginning of the LV motion estimation. It is the correspondence of SPAMM data over time that allows us to estimate the twisting motion of the LV.

5 Experiments

All our experiments run at interactive time speeds on a Silicon Graphics R4000 Crimson workstation, including the real time graphics. The set of data used in the following experiments were obtained from 10 image planes where five of them are from short-axis view planes and the other five are from the long-axis view planes. They span the spatial extent of a normal LV. Furthermore, for each image plane, we have data sets over five time sequences during systole (from end-diastole (time 1) to end-systole (time 5)). Therefore in total we have $5x5x5 = 75$ data sets of two-dimensional images, containing time-varying three-dimensional datapoints (boundary and SPAMM) of the LV.



(a) SPAMM datapoints from Short Axis View Image Planes

(b) SPAMM datapoints from Long Axis View Image Planes

Figure 8: SPAMM Data Sets

From each image, we extract 1) boundary datapoints, using snakes [7], from the inner and outer walls of the LV and, 2) SPAMM datapoints (i.e., intersections of tagging lines appear in the image) as shown in Fig. 9(a) and Fig. 8. For the experiments presented in this paper, we utilized

1. 265 outer boundary datapoints from the images (both short and long axes views) at time 1 (shown as black dots in Fig. 9(a)),

2. 261 inner boundary datapoints from the images at time 1 (shown as white dots in Fig. 9(a)),

3. SPAMM datapoints from 5 short axis view image planes over 5 time intervals (shown in Fig. 8(a)), where the number of datapoints[4] are 194, 174, 173, 166 and 157 at time 1, time

[4]Since some of the SPAMM datapoints on the image plane disappear and/or reappear at subsequent times, we used at every time frame only those points which have a corresponding point at the previous time frame. Therefore the number of *active* points decreases towards end-systole.

2, ... time 5, respectively.

4. SPAMM datapoints from 5 long axis image planes over 5 time intervals (shown in Fig. 8(b)). The number of datapoints are 216, 191, 182, 190, 182 at time 1, time 2, ... time 5, respectively.

It should be noted that we evaluate the spatial location of the image planes based on the acquired, during the SPAMM acquisition process, spatial locations of the corners of each image plane. We then express the coordinates of each datapoint wrt to the center of the LV.

5.1 Parameters

As described in Section 2.1, our deformable model is defined by 6 parameter functions which can be interpreted intuitively without any further complex processing. We first estimate the value of the material coordinate w for the inner and outer walls during fitting to the data in the first time frame (i.e., end-diastole). Then we estimate the parameter functions $a_1(u,w)$, $a_2(u,w)$ and $a_3(u,w)$ which are the model's aspect ratios along the x-, y- and z-axes, respectively. Since the short-axis views of the LV lie in the xy plane, the changes in $a_1(u,w)$ and $a_2(u,w)$ over time will capture the radial contraction of the LV. Likewise the changes in the aspect ratio along the z-axis (i.e., $a_3(u,w)$) will capture the longitudinal contraction of the LV. The estimated twisting parameter $\tau(u,w)$ is defined about the z-axis which coincides with the long axis of the LV. The final two estimated axis offset parameters $e_{1_o}(u,w)$ and $e_{2_o}(u,w)$ allow the long axis to be non-straight in the x and y directions, respectively. Thus we can capture more accurately the shape variation over time of the LV. Since all the above parameters vary with w, we can estimate their variation between the LV walls.

Table 1 summarizes what each parameter function, which is recovered during the fitting process, represents during the 3D shape and wall motion estimation of the LV.

Parameters	Representation
$a_1(u,w), a_2(u,w)$	magnitude of radial contractions
$a_3(u,w)$	magnitude of longitudinal contraction
$\tau(u,w)$	magnitude of twisting about the long axis
$e_{1_o}(u,w),\ e_{2_o}(u,w)$	magnitudes of long axis deformation

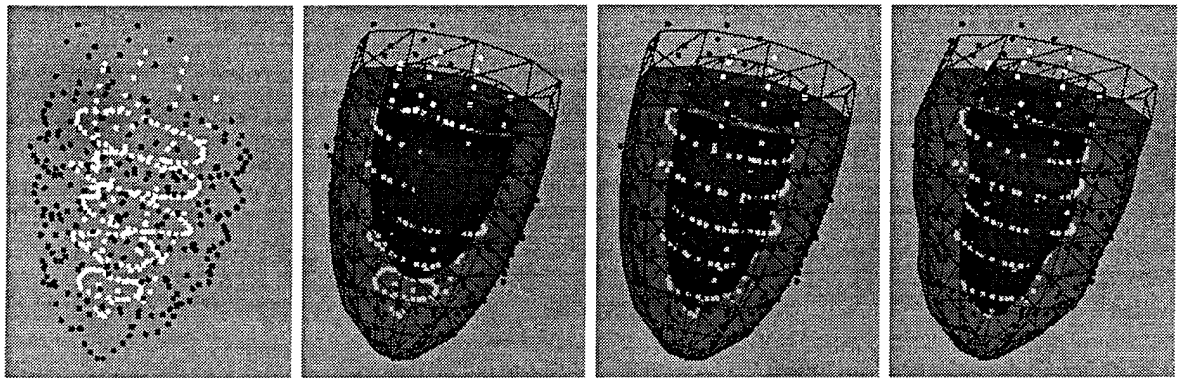
Table 1: Parameters

5.2 Model Fitting to Boundary datapoints at End-diastole

Based on the boundary datapoints from the inner and outer walls, we first recover the global shape of the LV at time 1 (i.e., end-diastole). This is done by overlaying the initial volumetric model onto the data. Then the nodes on the inner and outer surfaces of the model are pulled towards the

inner and outer boundary datapoints, respectively, based on the computation of forces described in Section 4.1. As a result of these forces the model parameter functions change so that the model conforms to the dataset. When all applied forces equilibrate or vanish, or the error of fit (the distance between a data point and the model surface) falls in an acceptable range, the model comes to rest.

The varying volume of the volumetric deformable model at the various stages of the initial fitting to the boundary datapoints is shown in Figs. 9(b-d). $w_{in} = 0.896$ and $w_{out} = 1.368$ for the model shown in Fig. 9(b). For better fitting results we initially allow a_1, a_2, a_3 to vary wrt to w (Fig. 9(c). The values of these parameters are: $a1(w_{out}) = 0.360$, $a1(w_{in}) = 0.282$, $a2(w_{out}) = 0.341$, $a2(w_{in}) = 0.276$, $a3(w_{out}) = 0.807$, and $a3(w_{in}) = 0.924$. We then allow all the parameters vary wrt to u, w (Fig. 9(d)).



(a) (b) (c) (d)

Figure 9: Initial shape recovery from boundary data.

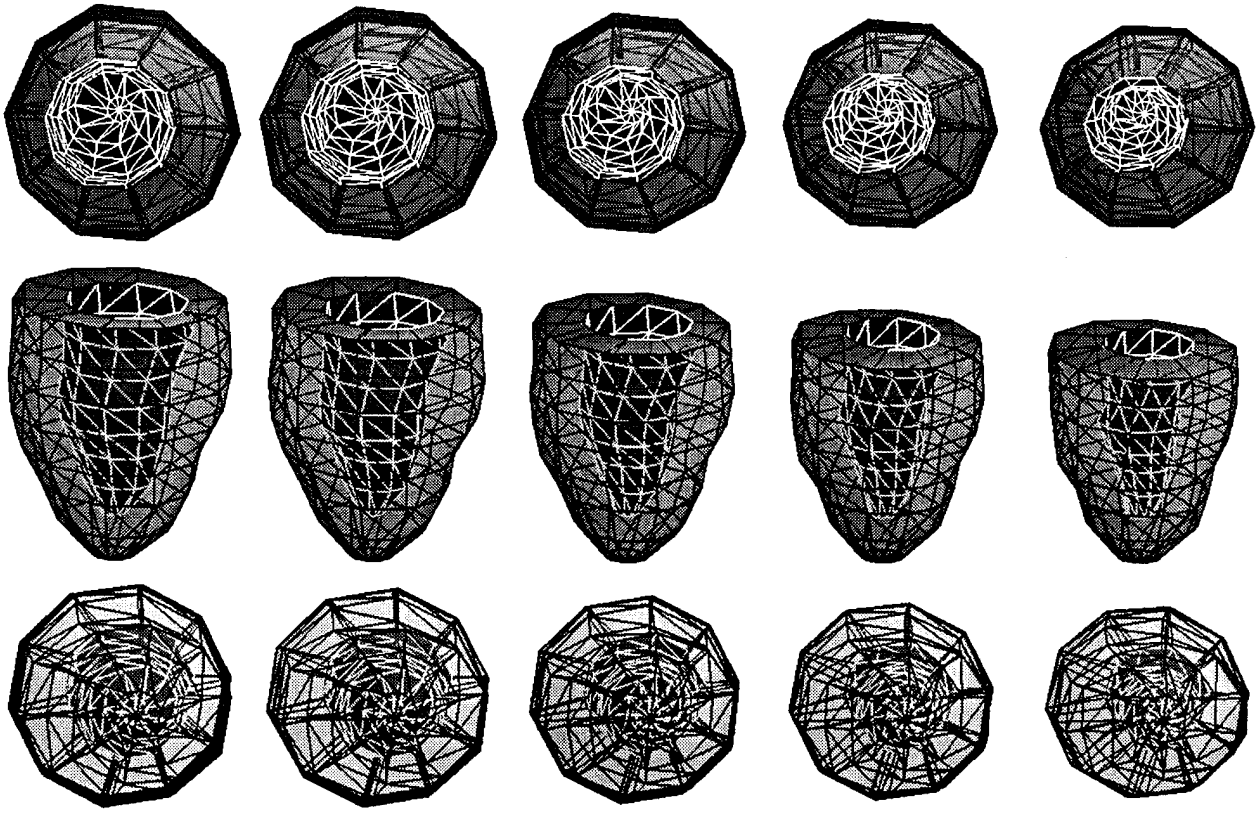
5.3 Model Fitting to Data Points over Time

Once we fit the model to the initial boundary data, the SPAMM points at the initial time are read and we find the corresponding model material coordinate points as describe in Section 4.2. During subsequent time frames we use both boundary and SPAMM data to fit the model. The force computation is done as described previously.

5.4 LV Fitting Results

Fig. 10 shows model fitting results over 5 time frames. The top row shows a view from the base of the LV of the fitted model. The twisting of the inner wall (shown in white) is obvious. The middle row shows a side view of the model, while the last row is similar to the first row and shows a view of the model from the apex. We can easily observe the longitudinal contraction as well as radial contraction.

Figs. 11(top,middle,bottom) show the fitted model superimposed to the SPAMM data at times $t = 1, 3, 5$, respectively. Columns (a),(b),(c) show the model with short axis SPAMM datapoints, long axis SPAMM datapoints and all the SPAMM datapoints, respectively. SPAMM datapoints



(a) time 1 (b) time 2 (c) time 3 (d) time 4 (d) time 5

Figure 10: Fitted models during systole

are denoted with black dots, while the corresponding model material points are denoted with white dots.

5.4.1 Analysis of the Estimation Results

Figs. 12(a-h) show graphs of the extracted model parameters as functions of u at the inner and outer walls. The difference in the corresponding parameter values is obvious. For example the radial contraction and the twisting are more significant in the inner LV wall (see Figs. 12(b,d,h)) compared to the outer walls (see Figs. 12(a,c,g)). The longitudinal contraction is similar in the inner and outer walls, but there is more contraction from the base to the apex. In the figures, the value 2.0 on the horizontal axis corresponds to the apex of the LV, while the value of 7.0 to a point close to the base. The above findings quantitatively verify qualitative knowledge common among physicians.

We can also measure the ejection fraction by computing volume changes of the inner cavity, observe changes in the myocardium thickness during systole, and extrapolate the parameter values throughout the muscle (in between walls) from our models.

Finally, in order to view the changes in the parameters during systole, we color-code the parameter variation of each parameter function along u and w on the SGI monitor display.

6 Conclusion

We have presented a new method for analyzing the 3D motion of the heart's left ventricle (LV) from MRI-SPAMM data. We developed a new class of volumetric physics-based deformable models whose parameters are functions. These parameters improve the accuracy of shape description through the use of a few intuitive parameters such as functional twisting. As opposed to the parameters of previous models for the LV, these parameters require no complex post-processing in order to be used by a physician. By developing new algorithms for estimating the forces from the SPAMM datapoints to the model's material points we were able to estimate the complex shape and motion of the LV. Through our analysis of the LV motion, we were able to quantitatively verify qualitative knowledge common among physicians. We plan to conduct experiments with several normal and abnormal hearts to establish the normal heart parameter variation.

References

[1] Axel L., Dougherty L., "Heart wall motion: Improved method of spatial modulation of magnetization for MR imaging," *Radiology*, 1989, **172**, pp. 349-350.

[2] Axel L., Goncalves F., Bloomgarden D., "Regional heart wall motion: Two-dimensional analysis and functional imaging of regional heart wall motion with magnetic resonance imaging," *Radiology*, 1992, **183**, pp.745-750.

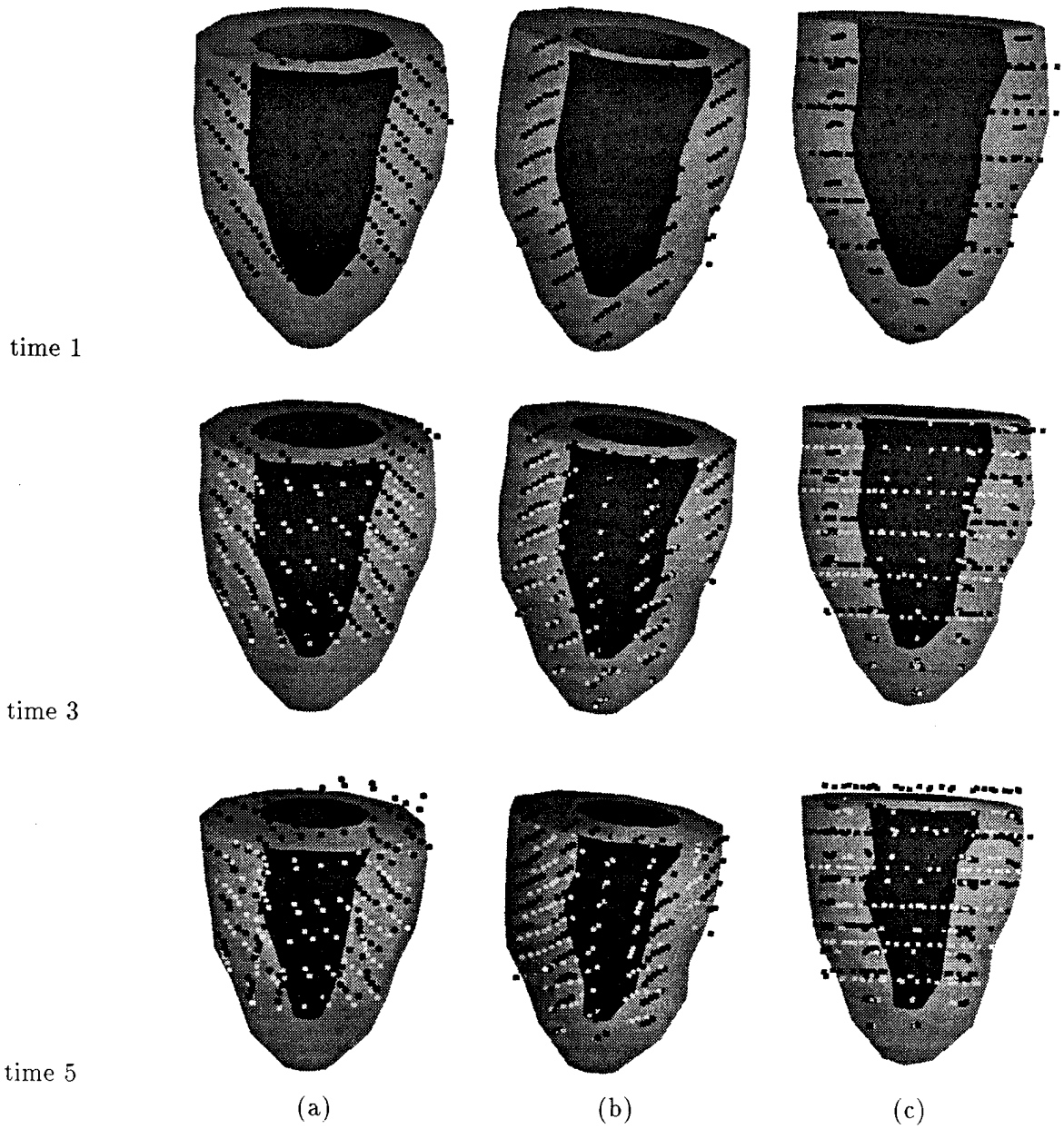


Figure 11: Fitted Models during systole (SPAMM datapoints and material points)

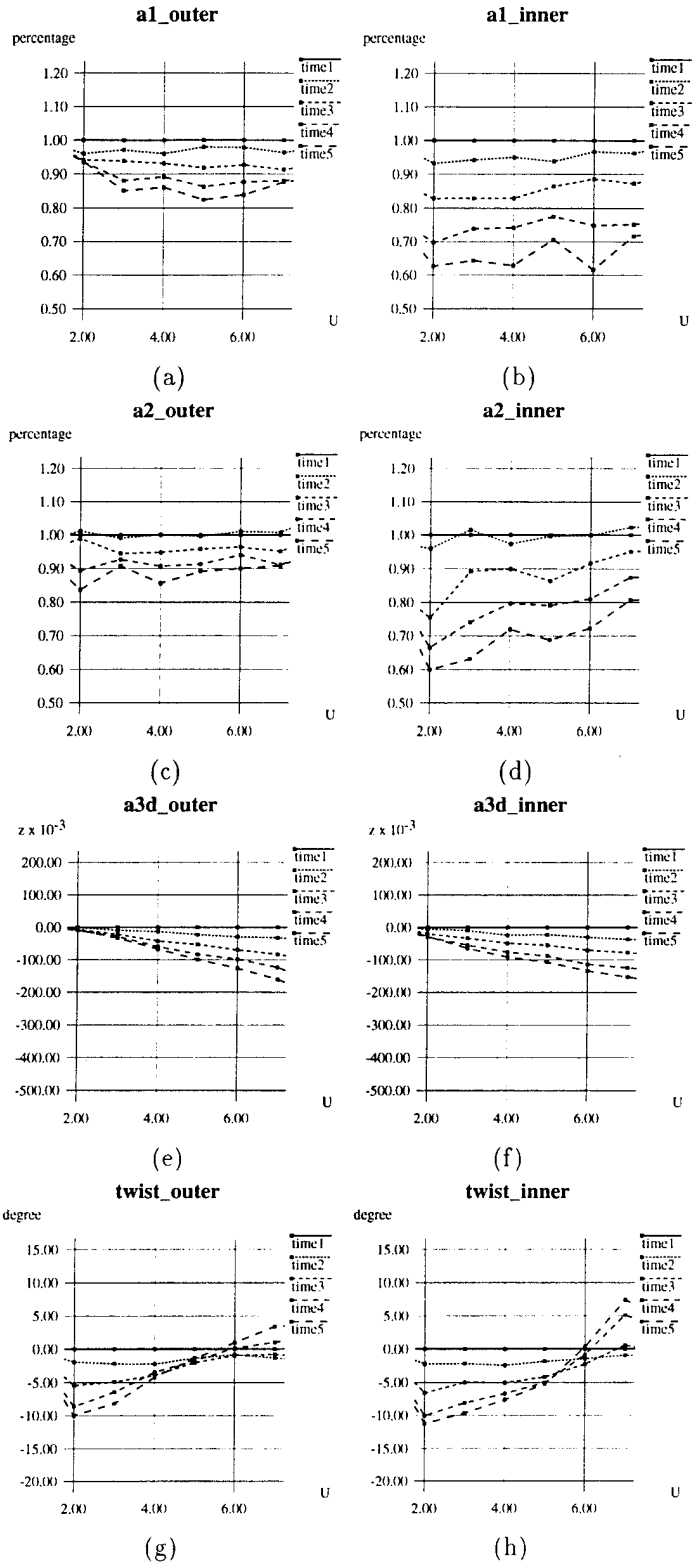


Figure 12: Extracted model parameters as functions of u at the inner and outer wall.

[3] Amini A., Duncan J., "Pointwise tracking of Left-Ventricular Motion in 3D," Proc. IEEE Workshop on Visual Motion, Princeton, NJ, 1991, pp. 294-298.

[4] Cohen L.D., Cohen I., "A Finite Element Method Applied to New Active Contour Models and 3D Reconstruction from Cross Sections," Proc. 2nd ICCV, Japan, 1990, pp. 587-591.

[5] Denney T.S. Jr., Prince J.L., "3D Displacement Filed Reconstruction on an Irregular Domain from Planar Tagged Cardiac MR Images," Proc. Workshop on Motion of Non-Rigid and Articulated Objects, pp. 172-177, Austin, TX, November 1994.

[6] Huang W.C., Goldgof D., "Adaptive-Size Meshes for Rigid and Nonrigid Shape Analysis and Synthesis," *IEEE Transactions on Pattern Analysis*, 1993, **15(6)**, pp. 611-616.

[7] Kass M., Witkin A., Terzopoulos D., "Snakes: Active Contour Models," *International Journal of Computer Vision*, 1988, **1(4)**, pp. 321-331.

[8] Marr D., Nishihara K., "Representation and Recognition of the Spatial Organization of Three-Dimensional Shapes," Proc. Royal Society London B, 1978.

[9] McInerney T., Terzopoulos D., "A Finite Element Model for 3D Shape Reconstruction and Nonrigid Motion Tracking," Proc. 4th International Conference on Computer Vision, Berlin, Germany, 1993, pp. 518-523.

[10] Metaxas D., Terzopoulos D., "Shape and Nonrigid Motion Estimation Through Physics-Based Synthesis", *IEEE Trans. Pattern Analysis and Machine Intelligence, 1993*, June, **15(6)**, pp. 569-579.

[11] Metaxas D., Koh E., "Flexible Multibody Dynamics and Adaptive Finite Element Techniques for Model Synthesis and Estimation," Proc. Second U.S. National Congress on Computational Mechanics, Washington, D.C., August, 1993.

[12] Moore C., O'Dell W., McVeigh E., Zerhouni E., "Calculation of three-dimensional left ventricular strains form biplanar tagged MR images," *J Mag Res Imag*, 1992, **2**, pp. 165-175.

[13] Nastar C., Ayache N., "Spatio-temporal analysis of nonrigid motion from 4D data," Proc. Workshop on Motion of Non-Rigid and Articulated Objects, pp. 146-151, Austin, TX, November 1994.

[14] Park J., Metaxas D., Young A., "Deformable Models with Parameter Functions: Application to Heart-Wall Modeling," Proc. IEEE Computer Vision and Pattern Recog. (CVPR'94), Seattle, WA, 1994, June, pp. 437-442.

[15] Park J., Metaxas D., Young A., Axel L., "Model-based Analysis of Cardiac Motion from Tagged MRI Data," Proc. Seventh Annual IEEE Symposium on Computer-Based Medical Systems, pp. 40-45, Winston-Salem, North Carolina, June 1994.

[16] Rogers W., Shapiro E., Weiss J., Buchalter M., Rademakers F., Weisfeldt M., Zerhouni E., "Quantification of and correction for left ventricular systolic long-axis shortening by magnetic resonance tissue tagging and slice isolation," *Circulation*, 1991, **84**, pp. 721-731.

[17] Shi P., Amini A., Robinson G., Sinusas A., Constable C.T., Duncan J., "Shape-based 4D Left Ventricular Myocardial Function Analysis," Proc. of IEEE Workshop on Biomedical Image Analysis, Seattle, WA, 1994, pp. 88-97.

[18] Pentland A., Horowitz B., "Recovery of Nonrigid Motion and Structure," *IEEE Pattern Analysis and Machine Intelligence*, 1991, July, **13(7)**, pp. 730-742.

[19] Terzopoulos D., Metaxas D., "Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics," *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1991, **13(7)**, pp.703–714.

[20] Villarreal F.L., Waldman L.K., Lew W.Y.W., "A Technique for measuring regional two-dimensional finite strains in cannie left ventricle," *Circ Res* 1988, **62**, pp. 711-721.

[21] Waldman L.K., Fung Y.C., Covell J.W., "Transmural myocardial deformation in the canine left ventrical: Normal in vivo three-dimensional finite strains," *Circ Res*, **57**, pp. 152-163, 1985.

[22] Young A.A., Hunter P.J., Smaill B.H., "Estimation of epicardial strain using the motions of coronary bifurcations in biplane cineangiography," *IEEE Trans Biomed Eng*, 1992, **39**, pp. 526-531.

[23] Young A.A., Axel L., "Three-dimensional Motion and Deformation of the Heart Wall," *Radiology*, 1992, **185**, pp. 241-247.

[24] Young A.A., Axel L., Dougherty L., Bogen D.K., Parenteau C.S., "Validation of Tagging with MR Imaging to Estimate Material Deformation," *Radiology*, 1993, **188**, pp. 101-108.

[25] Young A.A., Kraitchman D.L., Axel L., "Deformable Models for Tagged MR Images: Reconstruction of Two- and Three-Dimensional Heart Wall Motion," Proc. of IEEE Workshop on Biomedical Image Analysis, Seattle, WA, 1994, pp. 317-323.

[26] Zerhouni E.A., Parish D.M., Rogers W.J., Yang A., et al. "Human heart: Tagging with MR imaging - a method for noninvasive assessment of myocardial motion," *Radiology* 1988, **169**, pp. 59-63.

E Jack Reaching Planning With Strength Analysis and Collision Avoidance – User's Guide: Xinmin Zhao

Jack Reaching Planning With Strength Analysis and Collision Avoidance – User's Guide

Xinmin Zhao
Center for Human Modeling and Simulation
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd St. Philadelphia, Pennsylvania 19104-6389
Email: xzhao@graphics.cis.upenn.edu

February 15, 1995

1 Introduction

This document describes the implementation and usage a motion planning algorithm for human reaching motions. Given a goal position of the hand, and a description of the environment, this algorithm tries to find a collision free motion sequence that moves the hand to the goal position.

In general motion planning is a difficult problem. To plan the motion for an agent with n degrees of freedom, the complexity of the problem is exponential in n. For example, when n is 5, the problem can be approximated by constructing a grid of about $1E10$ nodes and finding a path in it. When there are many degrees of freedoms, such as the human reaching motion planning problem which involves at least 9 degrees of freedom, it is impractical to solve this problem completely.

There are many assumptions we can make to simplify the problem. For instance, we may assume the environment is 2D instead of 3D, or the obstacles are of certain simple shapes, etc. Here we adopt a different approach to

simplify the problem. Instead of guarantee a solution when there is one, this algorithm will find a solution efficiently in general, but it may fail to find a solution in some cases, even-though solutions may exist. And it works in 3D environment and can handle any shape of obstacles.

2 The algorithm

This algorithm is based on a robot motion planning algorithm developed by [1]. It is the fastest general algorithm available today. First, a 3D bitmap of the environment is constructed. In the bitmap 0s stand for empty (free) space while 1s stand for obstacles. Then the bitmap is searched to determine if there is a point path from the hand's start position to the goal position. If there is no point path, apparently there is no solution. If there is a point path, the algorithm proceed to find a motion sequence for the hand, arm and may be the body to move the hand to the goal position collision free.

This algorithm requires a 3D bitmap of the environment. Currently we use a bitmap of 64x64x64 (= 262144 nodes) to partition the environment. In order to have reasonable accuracy, the 3D space it covers cannot be too large. In the current implementation we limit the space to be a 200x200x200 cm^3 cube and assume the arm can only move inside the cube (the working cell of the arm). Since the bitmap is 64x64x64, the resolution of the partition is about 3cm, which is good enough considering human arm thickness. While it is possible to have larger bitmap sizes and cover more environment space, constructing and searching them will become very expensive.

Currently, the algorithm controls 9 degrees of freedom: one at the elbow joint, 3 at the shoulder, 3 at the waist, and 2 at the foot in the x and z direction (i.e., *Jack* can walk on the floor, but cannot climb up and down). For efficiency reason, it does not control the hip, knee, and ankle joints. It is assumed that *Jack* is in the "correct" or almost "correct" body posture before the reaching begins. *Jack* is limited to move on the floor no more than 32 cm in each direction.

3 New features since last release

There are two features added since last release: incorporating strength analysis into motion planning and providing a more informative user interface.

1. Strength Analysis:

   Using motion planning we can plan a collision free motion sequence to accomplish a given task. In some applications we also want to know if a person with limited strength is able to perform the planned motion. For this reason, we have incorporated strength analysis into our motion planning system. Given a sequence of planned motion, using inverse dynamics we compute the required torque at each joint in order to perform the motion. The required torques are compared against available torque data (collected by NASA) and determine if the motion is feasible. If a motion is not strength feasible, the program will point out which joint does not have enough strength to perform the planned motion.

2. Planning strength feasible motion:

   In order to planning a human task, finding a collision free motion may not be enough. Moreover, there are applications where we are interested in finding not only a collision free motion, but also a strength feasible motion as well. We have added another dimension to the motion planning problem: planning a motion which is both collision free and strength feasible for the particular agent whose strength data is given.

3. Improved user interface:

   In the current interface, the user is given more information on what's going on in the motion planning process. More specifically, the user is informed of:

   (a) The progress we are making towards the goal by displaying the distance of hand to the goal and the current posture of the agent;

   (b) How much time is left for the planner;

4 Commands

We provide the following commands under the menu "MotionPlan":

1. *init motion plan*

   This initializes the internal data structures for the motion planning algorithm. It should be executed before any other commands. It asks for a human figure and a space *reference site.*

   As mention in the previous section, this algorithm will limit the environment to be a 200x200x200 cm^3 cube. The *reference site* gives the origin of the cube. If the reference site has coordinates (x,y,z), then the two diagonal points on the cube are (x,y,z) and (x+200, y+200, z+200). The hand/arm motion will be limited to be inside the cube.

2. *input goal site*

   It asks for goal site name. Before each planning, the site's current location is used as the goal position.

3. *input obstacle*

   Input the obstacle segments.

   During the planning phase, collisions between the human arm/hand and the obstacles are detected at every step of the movement. To have good performance of the algorithm, it is essential to speed up the collision detection. In the current implementation, we do collision detection between:

   > palm, lower arm, upper arm and upper body with obstacles;
   >
   > palm, lower arm, upper arm with upper body (self collision detection).

   And all collision detections are done using bounding box collision detection. We ignore the collisions of fingers for the following reason:

   > It is time consuming to do. And the reaching can always be done by closing the hand.

4. *apply force on hand*

   This command inputs the external force acting on the hand (e.g., the weight of the object *Jack* is carrying). The user need to specify the acting point of the force (e.g., palm center site) and the x-y-z force vector (units in Newton). Initially all external forces are set to 0.

5. *plan a motion*

   This command computes the environment (obstacles) bitmap, evaluates the current start and goal positions, and computes the collision free motion sequences. It asks for the left hand or right hand, and the maximum allowed time to do the planning (in addition to the time used to compute the bitmap). The algorithm returns the best result it got if no solution is found when the time is up. (Note that it is a randomized algorithm, it may take different amount of time in each run, even for the same task. Typical running time ranges from seconds to minutes or more, depending on the task and the machine used.)

   This command only computes a collision free motion, and it does not consider strength in the planning process.

6. *plan a strength feasible motion*

   This command computes a motion sequence which is both collision free and strength feasible.

7. *strength analysis of the planned motion*

   This command analyzes a planned motion for strength feasibility. If the planned motion at one time exceed the available strength at a particular joint, the program will color that joint red. The joint will stay red until the the required strength is within the agent's strength limits.

8. *strength analysis of the current posture*

   This command checks if the required strength exceed the available strength of the agent in its current posture. Again, it colors red any joints that exceed the agent's strength.

9. *play the planned motion*

You may play the planned motion at slower speeds. This command asks for the playing speed. For example, input 2 means play the motion at 1/2 of the normal speed, while input 1 means play the planned motion at normal speed.

5 Motion Planning Examples

In the DEMO directory, there are two JCL files to test the motion planning algorithm. In the environment the transparent cube stands for the working cell of the hand. To the motion planning algorithm, outside the cube means obstacles.

The JCL files are: *B1_frame.jcl* and *B1_plan.jcl*. You may run the demo by issuing the flowing commands:

csh> *Depth*-Jack *B1_frame.jcl*

or

csh> *Depth*-Jack *B1_plan.jcl*

B1_frame.jcl will load the environment, and read the precomputed motion frames showing the planned reaching motion. *B1_plan.jcl* actually computes the motion frames used in *B1_frame.jcl*. (Note that this is a random algorithm. So the planned motion may be different each time the same task is planned.)

6 Examples with Strength Analysis

In the DEMO directory, there are two JCL files to show how feasible strength motion planning works. The JCL files are: *load_0.jcl* and *load_60.jcl*. Both shows motion planning of the same task: reaching for the top of the shelf. But in one case *Jack* is carry 60 Newtons in his hand, and in the other case *Jack* is carry 0 Newton.

References

[1] BARRAQUAND, J., AND LATOMBE, J.-C. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research 10*, 6 (December 1991), 628–649.

F Behavioral Control for Real-Time Simulated Human Agents: Granieri, Becket, Reich, Crabtree, and Badler

Behavioral Control for Real-Time Simulated Human Agents

John P. Granieri, Welton Becket,
Barry D. Reich, Jonathan Crabtree, Norman I. Badler

Center for Human Modeling and Simulation
University of Pennsylvania
Philadelphia, Pennsylvania 19104-6389
granieri/becket/reich/crabtree/badler@graphics.cis.upenn.edu

Abstract

A system for controlling the behaviors of an interactive human-like agent, and executing them in real-time, is presented. It relies on an underlying model of continuous *behavior*, as well as a discrete scheduling mechanism for changing behavior over time. A multiprocessing framework executes the behaviors and renders the motion of the agents in real-time. Finally we discuss the current state of our implementation and some areas of future work.

1 Introduction

As rich and complex interactive 3D virtual environments become practical for a variety of applications, from engineering design evaluation to hazard simulation, there is a need to represent their inhabitants as purposeful, interactive, human-like agents.

It is not a great leap of the imagination to think of a product designer creating a virtual prototype of a piece of equipment, placing that equipment in a virtual workspace, then populating the workspace with virtual human operators who will perform their assigned tasks (operating or maintaining) on the equipment. The designer will need to instruct and guide the agents in the execution of their tasks, as well as evaluate their performance within his design. He may then change the design based on the agents' interactions with it.

Although this scenario is possible today, using only one or two simulated humans and scripted task animations [3], the techniques employed do not scale well to tens or hundreds of humans. Scripts also limit any ability to have the human agents react to user input as well as each other during the execution of a task simulation. We wish to build a system capable of simulating many agents, performing moderately complex tasks, and able to react to external (either from user-generated or distributed simulation) stimuli and events, which will operate in near real-time. To that end, we have put together a system which has the beginnings of these attributes, and are in the process of investigating the limits of our approach. We describe below our architecture, which employs a variety of known and previously published techniques, combined together in a new way to achieve near real-time behavior on current workstations.

We first describe the machinery employed for behavioral control. This portion includes perceptual, control, and motor components. We then describe the multiprocessing framework built to run the behavioral system in near real-time. We conclude with some internal details of the execution environment. For illustrative purposes, our example scenario is a pedestrian agent, with the ability to locomote, walk down a sidewalk, and cross the street at an intersection while obeying stop lights and pedestrian crossing lights.

2 Behavioral Control

The behavioral controller, previously developed in [4] and [5], is designed to allow the operation of parallel, continuous behaviors each attempting to accomplish some function relevant to the agent and each connecting sensors to effectors. Our behavioral controller is based on both potential-field reactive control from robotics [1, 10] and behavioral simulation from graphics, such as Wilhelms and Skinner's implementation [20] of Braitenberg's *Vehicles* [7]. Our system is structured in order to allow the application of optimization learning [6], however, as one of the primary difficulties with behavioral and reactive techniques is the complexity of assigning weights or arbitration schemes to the various behaviors in order to achieve a desired observed behavior [5, 6].

Behaviors are embedded in a network of *behavioral nodes*, with fixed connectivity by links across which only floating-point messages can travel. On each simulation step the network is updated synchronously and without order dependence by using separate load and emit phases using a simulation technique adapted from [14]. Because there is no order dependence, each node in the network could be on a separate processor, so the network could be easily parallelized.

Each functional behavior is implemented as a subnetwork of behavioral nodes defining a path from the geometry database of the system to calls for changes in the database. Because behaviors are implemented as networks of simpler processing units, the representation is more explicit than in behavioral controllers where entire behaviors are implemented procedurally. Wher-

ever possible, values that could be used to parameterize the behavior nodes are made accessible, making the entire controller accessible to machine learning techniques which can tune components of a behavior that may be too complex for a designer to manage. The entire network comprising the various sub-behaviors acts as the controller for the agent and is referred to here as the *behavior net*.

There are three conceptual categories of behavioral nodes employed by behavioral paths in a behavior net:

perceptual nodes that output more abstract results of perception than what raw sensors would emit. Note that in a simulation that has access to a complete database of the simulated world, the job of the perceptual nodes will be to realistically limit perception, which is perhaps opposite to the function of perception in real robots.

motor nodes that communicate with some form of motor control for the simulated agent. Some motor nodes enact changes directly on the environment. More complex motor behaviors, however, such as the *walk motor node* described below, schedule a motion (a step) that is managed by a separate, asynchronous execution module.

control nodes which map perceptual nodes to motor nodes usually using some form of negative feedback.

This partitioning is similar to Firby's partitioning of continuous behavior into active sensing and behavior control routines [10], except that motor control is considered separate from negative feedback control.

2.1 Perceptual Nodes

The perceptual nodes rely on simulated sensors to perform the perceptual part of a behavior. The sensors access the environment database, evaluate and output the distance and angle to the target or targets. A sampling of different sensors currently used in our system is described below. The sensors differ only in the types of things they are capable of detecting.

Object: An object sensor detects a single object. This detection is global; there are no restrictions such as visibility limitations. As a result, care must be taken when using this sensor: for example, the pedestrian may walk through walls or other objects without the proper avoidances, and apparent realism may be compromised by an attraction to an object which is not visible. It should be noted that an object sensor always senses the object's current location, even if the object moves. Therefore, following or pursuing behaviors are possible.

Location: A location sensor is almost identical to an object sensor. The difference is that the location is a unchangeable point in space which need not correspond to any object.

Proximity: A proximity sensor detects objects of a specific type. This detection is local: the sensor can detect only objects which intersect a sector-shaped region roughly corresponding to the field-of-view of the pedestrian.

Line: A line sensor detects a specific line segment.

Terrain: A terrain sensor, described in [17], senses the navigability of the local terrain. For example, the pedestrian can distinguish undesirable terrain such as street or puddles from terrain easier or more desirable to negotiate such as sidewalk.

Field-of-View: A field-of-view sensor, described in [17], determines whether a human agent is visible to any of a set of agents. The sensor output is proportional to the number of agents' fields-of-view it is in, and inversely proportional to the distances to these agents.

2.2 Control Nodes

Control nodes typically implement some form of negative feedback, generating outputs that will reduce perceived error in input relative to some desired value or limit. This is the center of the reactivity of the behavioral controller, and as suggested in [9], the use of negative feedback will effectively handle noise and uncertainty.

Two control nodes have been implemented as described in [4] and [5], *attract* and *avoid*. These loosely model various forms of *taxis* found in real animals [7, 11] and are analogous to proportional servos from control theory. Their output is in the form of a recommended new velocity in polar coordinates:

Attract An *attract* control node is linked to θ and d values, typically derived from perceptual nodes, and has angular and distance thresholds, t_θ and t_d. The *attract* behavior emits $\Delta\theta$ and Δd values scaled by linear weights that suggest an update that would bring d and θ closer to the threshold values. Given weights k_θ and k_d :

$$\Delta\theta = \begin{cases} 0 & \text{if } -t_\theta \leq \theta \leq t_\theta \\ k_\theta(\theta - t_\theta) & \text{if } \theta > t_\theta \\ k_\theta(\theta + t_\theta) & \text{otherwise} \end{cases}$$

$$\Delta d = \begin{cases} 0 & \text{if } d \leq t_d \\ k_d(d - t_d) & \text{otherwise.} \end{cases}$$

Avoid The *avoid* node is not just the opposite of *attract*. Typically in *attract*, both θ and d should be within the thresholds. With *avoid*, however, the intended behavior is usually to have d outside the threshold distance, using θ only for steering away. The resulting avoid formulation has no angular threshold:

$$\Delta\theta = \begin{cases} 0 & \text{if } d > t_d \\ k_\theta(\pi - \theta) & \text{if } d \leq t_d \text{ and } \theta \geq 0 \\ k_\theta(-\pi - \theta) & \text{otherwise} \end{cases}$$

$$\Delta\theta = \begin{cases} 0 & \text{if } d > t_d \\ k_d(t_d - d) & \text{otherwise.} \end{cases}$$

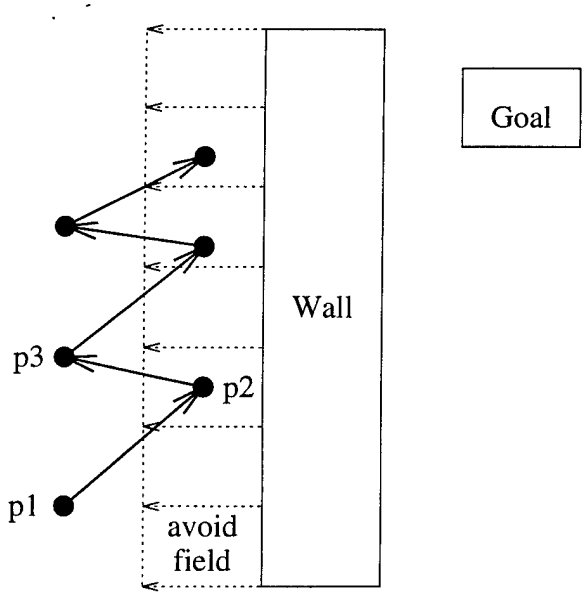


Figure 1: Sawtooth path due to potential field discontinuities

2.3 Motor Nodes

Motor nodes for controlling non-linked agents are implemented by interpreting the Δd and $\Delta\theta$ values emitted from control behaviors as linear and angular adjustments, where the magnitude of the implied velocity vector gives some notion of the urgency of traveling in that direction. If this velocity vector is attached directly to a figure so that requested velocity is mapped directly to a change in the object's position, the resulting agent appears jet-powered and slides around with infinite damping as in Wilhelms and Skinner's environment [20].

2.3.1 Walking by sampling potential fields

When controlling agents that walk, however, the motor node mapping the velocity vector implied by the outputs of the control behaviors to actual motion in the agent needs to be more sophisticated. In a walking agent the motor node of the behavior net *schedules* a step for an agent by indicating the position and orientation of the next footstep, where this decision about where to step next happens at the end of every step rather than continuously along with motion of the agent. The velocity vector resulting from the blended output of all control nodes could be used to determine the next footstep; however, doing so results in severe instability around threshold boundaries. This occurs because we allow thresholds in our sensor and control nodes and as a result the potential field space is not continuous. Taking a discrete step based on instantaneous information may step across a discontinuity in field space. Consider the situation in Fig. 1 where the agent is attracted to a goal on the opposite side of a wall and avoids the wall up to some threshold distance. If the first step is scheduled at position p_1, the agent will choose to step directly toward the goal and will end up at p_2. The agent is then well within the threshold distance for walls and will step away from the wall and end up at p_3, which is outside the threshold. This process then repeats until the wall

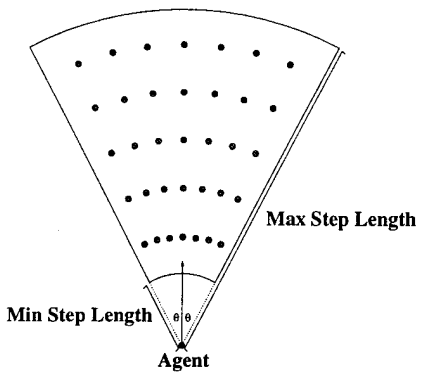


Figure 2: The fan of potential foot locations and orientations

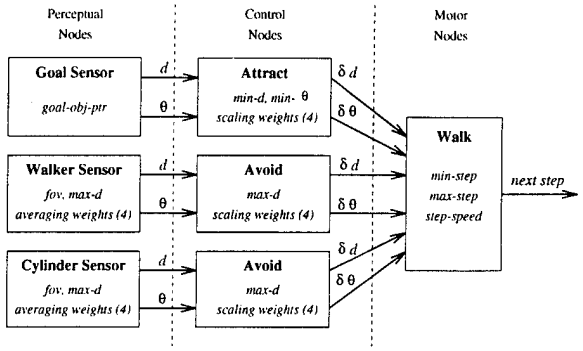


Figure 3: An example behavior net for walking

is cleared, producing an extremely unrealistic sawtooth path about the true gradient in the potential field.

To eliminate the sawtooth path effect, we sample the value of the potential field implied by the sensor and control nodes in the space in front of the agent and step on the location yielding the minimum sampled 'energy' value. We sample points that would be the agent's new location if the agent were to step on points in a number of arcs within a fan in front of the agent's forward foot. This fan, shown in Fig. 2, represents the geometrically valid foot locations for the next step position under our walking model. This sampled step space could be extended to allow side-stepping or turning around which the agent can do [3], though this is not currently accessed from the behavior system described in this paper. For each sampled step location, the potential field value is computed at the agent's new location, defined as the average location and orientation of the two feet.

2.4 An example behavior net

The example behavior net in Fig. 3 specifies an overall behavior for walking agents that head toward a particular goal object while avoiding obstacles (cylinders in this case) and each other. The entire graph is the *behavior net*, and each path from perception to motor output is considered a *behavior*. In this example there are three behaviors: one connecting a goal sensor to an attraction controller and then to the walk node (a goal-attraction behavior), another connecting a sensor detecting proximity of other walking agents to an avoidance controller

and then to the walk node (a walker-avoidance behavior), and a final behavior connecting a cylinder proximity sensor to an avoidance behavior and then to the walk node (a cylinder-avoidance behavior).

Each node has a number of parameters that determine its behavior. For example, the walker sensor and the cylinder sensor nodes have parameters that indicate how they will average all perceived objects within their field of view and sensing distance into a single abstract object. The Attract and Avoid nodes have scaling weights that determine how much output to generate as a function of current input and the desired target values.

The walk motor behavior manages the sampling of the potential field by running data through the perceptual and control nodes with the agent pretending to be in each of the sampled step locations. The walk node then schedules the next step by passing the step location and orientation to the execution module.

Note that this example has no feedback, cross-talk, or inhibition within the controller, though the behavioral controller specification supports these features [5]. Although this example controller itself is a feed-forward network, it operates as a closed-loop controller when attached to the agent because the walk node's scheduling of steps affects the input to the perceptual nodes.

Our use of *attract* and *avoid* behaviors to control groups of walking agents may appear on the surface like Ridsdale's use of *hot* and *cold* tendencies to control agents in his *Director's Apprentice* system [18]. However, his system was not reactive and on-line as our behavioral controller is, it did not limit perception of agents, it had no structured facilities for tuning behavior parameters, and it did not take advantage of developments in reactive control and behavioral simulation. His system focused on the use of an expert system to schedule human activity conforming to stage principles and used hot and cold tendencies to manage complex human behavior and interaction. We limit the use of behaviors to reactive navigation and path-planning, using parallel transition networks rather than one large expert system to schedule events, and we look to symbolic planning systems based on results in cognitive science, such as [3, 8, 16], to automate high-level human behavior and complex human interactions.

3 Parallel Automata

Parallel Transition Networks (PaT-Nets) are transition networks that run in parallel with the behavior net, monitor it, and edit it over time [8]. They are a mechanism for scheduling arbitrary actions and introducing decision-making into the agent architecture. They monitor the behavior net (which may be thought of as modeling low level instinctive or reflexive behavior) and make decisions in special circumstances. For example, the agent may get caught in a dead-end or other local minimum. PaT-Nets recognize situations such as these, override the "instinctive" behavior simulation by reconfiguring connectivity and modifying weights in the behavior net, and then return to a monitoring state.

In our pedestrian example we combine object and location sensors (in perceptual nodes) with *attract* control nodes, and proximity and line sensors (in perceptual nodes) with *avoid* control nodes. Pedestrians are attracted to street corners and doors, and they avoid each other, light poles, buildings, and the street except at crosswalks.

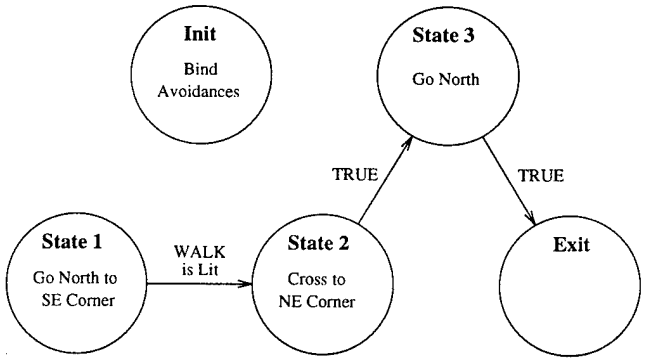


Figure 4: `North-net`: A sample `ped-net` shown graphically

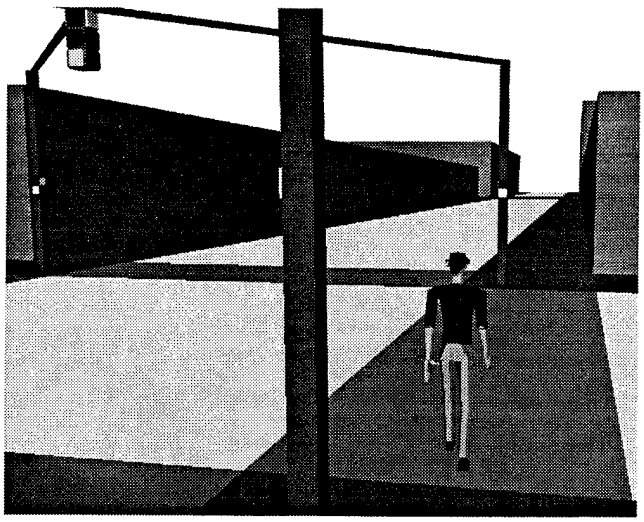


Figure 5: A pedestrian crossing the street

We use PaT-Nets in several different ways. `Light-nets` control traffic lights and `ped-nets` control pedestrians. `Light-nets` cycle through the states of the traffic light and the *walk* and *don't walk* signs.

Fig. 4 is a simple `ped-net`, a `north-net`, which moves a pedestrian north along the eastern sidewalk through the intersection. Initially, avoidances are bound to the pedestrian so that it will not walk into walls, the street, poles, or other pedestrians. The avoidances are always active even as other behaviors are bound and unbound. In State 1 an attraction to the southeast corner of the intersection is bound to the pedestrian. The pedestrian immediately begins to walk toward the corner avoiding obstacles along the way. When it arrives the attraction is unbound, the action for State 1 is complete. Nothing further happens until the appropriate walk light is lit. When it is lit, the transition to State 2 is made and action *Cross to NE Corner* is executed. The agent crosses the street. Finally, the agent heads north.

Fig. 5 shows a pedestrian controlled by a `north-net`. The transition to State 2 was just made so the pedestrian is crossing the street at the crosswalk.

4 Real-Time Simulation Environment

The run-time simulation system is implemented as a group of related processes, which communicate through shared memory. The system is broken into a minimum of 5 processes, as shown in Fig. 6. The system relies on IRIS Performer [19] for the general multiprocessing framework. Synchronization of all processes, via spin locks and video clock routines, is performed in the CONTROL process. It is also the only process which performs the edits and updates to the run-time visual database. The CULL and DRAW processes form a software rendering pipeline, as described in [19]. The pipeline improves overall rendering throughput while increasing latency, although the two frame latency between CONTROL and DRAW is not significant for our application. Our CONTROL process is equivalent to the APP process in the Performer framework. We have used this framework to animate multiple real-time human figures [12].

4.1 CONTROL Process

The CONTROL process runs the main simulation loop for each agent. This process runs the PaT-Nets, and underlying behavior net for each agent. While each agent has only one behavior net, they may have several PaT-Nets running, which sequence the parameters and connectivity of the nodes in the behavior net over time (as shown in Fig. 6).

By far the costliest computation in the CONTROL process, for the behaviors modeled in this example application, is the evaluation of the **Walk** motor node in the behavior net, and specifically the selection of the next foot position. Since this computation is done only once for every footfall, it usually runs only every 15 frames or so (the average step time being about 1/2 second, and average frame rate 30Hz). If the CONTROL process starts running over its allotted frame time, the **Walk** nodes will start reducing the number of points sampled for the next foot position, thereby reducing computation time. The only danger here is described in Section 2.3.1, the potential for a sawtooth path. If many agents are walking at similar velocities, they can all end up computing their next-step locations at the same frame-time, creating a large computation spike which causes the whole simulation to hiccup. (It is visually manifested by the feet landing in one frame, then the swing foot suddenly appearing in mid-stride on the next frame.) We attempt to even out the computational load for the **Walk** motor node evaluation by staggering the start times for each agent, and thereby distributing the computation over about 1/2 second for all agents.

Another computational load in the CONTROL process comes from the evaluation of the conditional expressions in the Pat-Nets, which may occur on every frame of the simulation. They are currently implemented via LISP expressions, so evaluating a condition involves parse and eval steps. In practice, this is fairly fast as we precompile the LISP, but as the PaT-Nets increase in complexity it will be necessary to replace LISP with a higher performance language (i.e. compiled C code). This may remove some of the generality and expressive power enjoyed with LISP.

Another technique employed to improve performance, when evaluating a large number of Pat-Nets and behavior nets, is to have the CONTROL process spawn copies of itself, with each copy running the behavior of a subset of the agents. This works as long as updates to the visual database are exclusive to each CONTROL process. (In practice this is the case, since the current behavior net for one agent will not edit any parameters for another agent in the visual database.) Of course, the assumption in spawning more processes is that there are available CPUs to run them.

The CONTROL process also provides the outputs of the motor nodes in the behavior net to the MOTION process. These outputs, in the case of the walking behavior, are the position and orientation of the agent's next foot fall. It also evaluates the motion data (joint angles) coming from the MOTION process, and performs the necessary updates to the articulation matrices of the human agent in the visual database.

4.2 SENSE Process

The SENSE process controls and evaluates the simulated sensors modeled in the perceptual nodes of the behavior net. It provides the outputs of the perceptual nodes to the CONTROL process, which uses them for the inputs to the control nodes of the behavior net. The main computational mechanism the sensors employ are intersections of simple geometric shapes (a set of points, lines, frustums or cones) with the visual database, as well as distance computations. This process corresponds to an ISECT process in the Performer framework.

The major performance parameters of this process are the total number of sensors as well as the complexity and organization of the visual database. Since it needs read-only access to the visual database, several SENSE processes may be spawned to balance the load between the number of sensors being computed, and the time needed to evaluate them. (These extra processes are represented by the dotted SENSE process in Fig. 6.) There is a one frame latency between the outputs of the perceptual nodes and the inputs to the control nodes in the behavior net (which are run in the CONTROL process), but this is not a significant problem for our application.

4.3 MOTION Process

Once the agent has sensed its environment and decided on on appropriate action to take, its motion is rendered via real-time motion generators, using a motion system that mixes pre-recorded playback and fast motion generation techniques.

We use an off-line motion authoring tool [2, 13] to create and record motions for our human figures. The off-line system organizes motion sequences into *posture graphs* (directed, cyclic graphs). Real-time motion playback is simply a traversal of the graph in time. This makes the run-time motion generation free from frame-rate variations. The off-line system also records motions for several levels-of-detail (LOD) models of the human figure. (Both the bounding geometry of the figure, as well as the articulation hierarchy (joints) are represented at several levels of detail.) The three levels-of-detail we are using for the human figure are:

1. A 73 joint, 130 DOF, 2000 polygon model, which has articulated fingers and flexible torso, for use in close-up rendering, and fine motor tasks (*Jack*®),

2. A 17 joint, 50 DOF, 500 polygon model, used for the bulk of rendering; it has no fingers, and the flexible torso has been replaced by two joints,

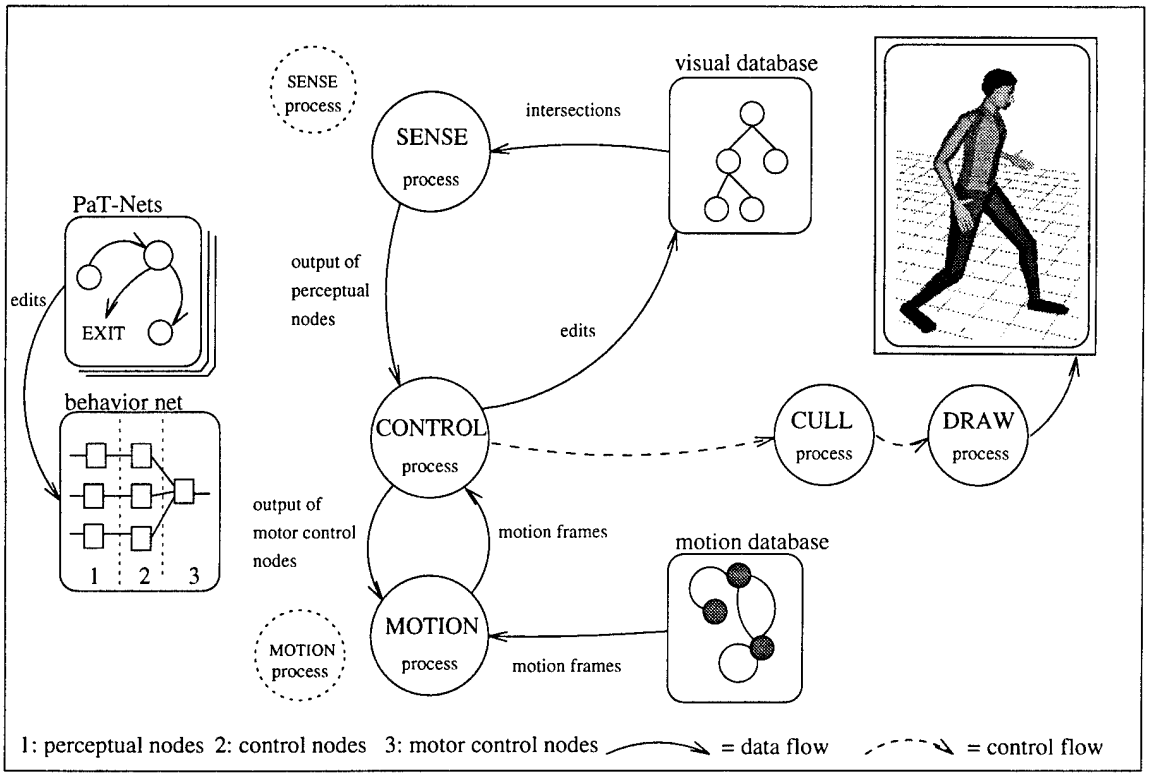


Figure 6: The multiprocessing framework for the real-time behavior execution environment

3. An 11 joint, 21 DOF, 120 polygon model used when the human agent is at a large distance from the camera.

This process produces a frame of motion for each agent, then sleeps until the next frame boundary (the earliest any new motion could be needed). It provides the correct motion frame for the currently active LOD model in the visual database. For certain types of sensors modeled in the perceptual nodes, this process will also be requested to provide a full (highest LOD) update to the visual database, in the case where a lower LOD is currently being used, but a sensor needs to interact with the highest LOD model.

The motion database consists of one copy of the posture graphs and associated motion between nodes of the posture graph. Each transition is stored at a rate of 60HZ, on each LOD model of the human agent. This database is shared by all agents. Only a small amount of private state information is maintained for each agent.

The MOTION process can effectively handle about 10-12 agents at update rates of 30Hz (on a 100MHz MIPS R4000 processor). Since the process only has read-only access to the motion database, we can spawn more MOTION processes if needed for more agents.

4.4 Walking as an example

A MOTION process animates the behaviors specified by an agent's motor nodes by playing back what are essentially pre-recorded chunks of motion. As a time-space tradeoff, this technique provides faster and less variable run-time execution at the cost of additional storage requirements and reduced generality. The interesting issues arise in how we choose a mapping from motor node outputs to this discrete representation; it plays a significant role in determining how realistic the animated agents will be.

The primary motor behavior to be executed is walking. Our full walking algorithm combines kinematics with dynamic balance control and is capable of generating arbitrary curved-path locomotion [15]. In order to reduce computational costs, however, we have not incorporated the algorithm directly into our run-time system. Instead, as implied by the preceding discussion, we record canonical "left" and "right" steps generated by the algorithm (which **is** a component of our off-line motion authoring system) and then play them back in an alternating fashion to produce a continuous walking motion.

The input to the appropriate MOTION process's walking subsystem consists of the specification of the desired next foot position and orientation (for the swing foot). This input is itself already discretized, as the motor node responsible (the **Walk** motor node) for evaluating how desirable it is for the agent to be at particular positions only computes the desirability criteria at a set number of points (in Fig. 2). However, even given that there are only n possibilities for the placement of the swing foot on the next step, this would still require us to record order n^2 possible steps, since the planted foot could be in any one of the n different positions at the start of the step (determined by the **last** step taken) and any one of the n at the end.

Without recording all n^2 distinct steps it is necessary to choose the best match among those that we do record. One of the most important criteria in obtaining realistic results is to minimize foot slippage relative to

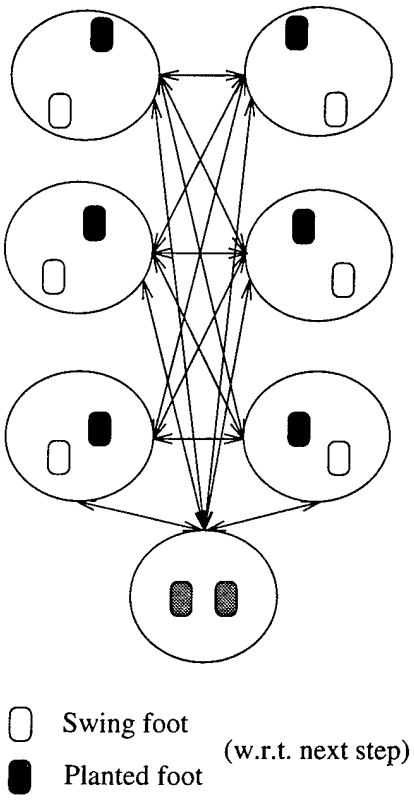


Figure 7: Posture graph for variable step length walking (3 step sizes)

the ground; foot slippage occurs when the pre-recorded movement (in particular its amount and direction) does not match that specified by the **walk** motor node at run time. On the basis that translational foot slippage is far more evident than rotational slippage (at least from our informal observations), we currently adopt an approach in which we record three types of step: short, medium, and long. Turning is accomplished by rotating the agent around his planted foot smoothly throughout the step. Having three step sizes significantly increases the chances of being able to find a close match to the desired step size, and, in fact, the **walk** motor node can be constrained to **only** consider the three arcs of the next foot location fan (see Fig. 2) that correspond exactly to our recorded step sizes. Doing so eliminates translational slippage, but has the sawtooth hazard.

The posture graph for all possible step-to-step transitions is shown in Fig. 4.4. Notice that even with only three kinds of straight-line walking there are many possible transitions, and hence numerous motion segments to be recorded. However, allowing for variable step length is very important. For instance, an attract control node can be set to drive the agent to move within a certain distance of a goal location; were there only a single step size, the agent might be unable to get sufficiently close to the goal without overshooting it each time, resulting in degenerate behavior (and possible virtual injury).

One thing worthy of mention with respect to the number of different walking steps required to reproduce arbitrary curved-path locomotion is that while there are theoretically order n^2 of them, the similarities are significant. It is thus possible that it will prove feasible to store a single full set of steps along with a little more information to represent how those steps can be modified slightly to realistically turn the agent left or right, and make it sufficiently fast for our real-time applications.

5 Conclusions and Future Work

We have designed a multiprocessing system for the real-time execution of behaviors and motions for simulated human-like agents. We have used only toy examples to date, and are eager to push the limits of the system to model more complex environments and interactions amongst the agents.

Although our agents currently have limited abilities (locomotion and simple posture changes), we will be developing the skills for interactive agents to perform maintenance tasks, handle a variety of tools, negotiate terrain, and perform tasks in cramped spaces. Our goal is a system which does not provide for all possible behaviors of a human agent, but allows for new behaviors and control techniques to be added and blended with the behaviors and skills the agent already possesses.

We have used a coarse grain parallelism to achieve interactive frame rates. The behavior net lends itself to finer grain parallelism, as one could achieve using a threaded approach. Our system now is manually tuned and balanced (between the number of agents, the number of sensors per agent, and the complexity of the visual database). A fruitful area of research is in the automatic load balancing of the MOTION and SENSE processes, spawning and killing copies of these processes, and doling out agents and sensors, as agents come and go in the virtual environment. Results in real-time system scheduling and approximation algorithms will be applicable here.

6 Acknowledgments

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory; Naval Training Systems Center N61339-93-M-0843; Sandia Labs AG-6076; ARPA AASERT DAAH04-94-G-0362; DMSO DAAH04-94-G-0402; ARPA DAMD17-94-J-4486; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; DMSO through the University of Iowa; and NSF CISE CDA88-22719.

References

[1] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 105–122. MIT Press, 1990.

[2] 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.

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

[4] Welton Becket. *Simulating Humans: Computer Graphics, Animation, and Control*, chapter Controlling forward simulation with societies of behaviors.

[5] Welton Becket and Norman I. Badler. Integrated behavioral agent architecture. In *The Third Conference on Computer Generated Forces and Behavior Representation*, Orlando, Florida, March 1993.

[6] Welton M. Becket. *Optimization and Policy Learning for Behavioral Control of Simulated Autonomous Agents.* PhD thesis, University of Pennsylvania, 1995. In preparation.

[7] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology.* The MIT Press, 1984.

[8] J. Cassell, C. Pelachaud, N. Badler, M. Steedman, B. Achorn, W. Becket, B. Douville, S. Prevost, and M. Stone. Animated conversation: rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *Proceedings of SIGGRAPH '94. In Computer Graphics*, pages 413–420, 1994.

[9] Thomas L. Dean and Michael P. Wellman. *Planning and Control.* Morgan Kaufmann Publishers, Inc., 1991.

[10] R. James Firby. Building symbolic primitives with continuous control routines. In *Artificial Intelligence Planning Systems*, 1992.

[11] C. R. Gallistel. *The Organization of Action: A New Synthesis.* Lawrence Elerbaum Associates, Publishers, Hillsdale, New Jersey, 1980. Distributed by the Halsted Press division of John Wiley & Sons.

[12] John P. Granieri and Norman I. Badler. In Ray Earnshaw, John Vince, and Huw Jones, editors, *Applications of Virtual Reality*, chapter Simulating Humans in VR. Academic Press, 1995. To appear.

[13] John P. Granieri, Johnathan Crabtree, and Norman I. Badler. Off-line production and real-time playback of human figure motion for 3d virtual environments. In *IEEE Virtual Reality Annual International Symposium*, Research Triangle Park, NC, March 1995. To appear.

[14] David R. Haumann and Richard E. Parent. The behavioral test-bed: obtaining complex behavior from simple rules. *The Visual Computer*, 4:332–337, 1988.

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

[16] Micheal B. Moore, Christopher W. Geib, and Barry D. Reich. Planning and terrain reasoning. In *Working Notes - 1995 AAAI Spring Symposium on Integrated Planning Applications.*, 1995. to appear.

[17] Barry D. Reich, Hyeongseok Ko, Welton Becket, and Norman I. Badler. Terrain reasoning for human locomotion. In *Proceedings of Computer Animation '94*, Geneva, Switzerland, May 1994. IEEE Computer Society Press.

[18] Gary Ridsdale. *The Director's Apprentice: Animating Figures in a Constrained Environment.* PhD thesis, Simon Fraser University, School of Computing Science, 1987.

[19] John Rohlf and James Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. *Computer Graphics*, pages 381–394, 1994.

[20] Jane Wilhelms and Robert Skinner. A 'notion' for interactive behavioral animation control. *IEEE Computer Graphics and Applications*, 10(3):14–22, May 1990.

G Planning and Terrain Reasoning: Moore, Geib, and Reich

Planning and Terrain Reasoning*

Michael B. Moore Christopher Geib Barry D. Reich

University of Pennsylvania
Department of Computer and Information Science
200 S. 33rd Street, Philadelphia, PA 19104-6389

E-mail: {mmoore,geib,reich}@graphics.cis.upenn.edu

Abstract

We describe the ZAROFF system, a plan-based controller for the player who is "it" in a game of hide and seek. The system features visually realistic human figure animation including realistic human locomotion. We discuss the planner's interaction with a changing environment to which it has limited perceptual access.

Introduction

The game of *hide and seek* challenges the ability of players to plan for acquiring information. The player who is "it" (hereafter called the *seeker*) must explore his environment attempting to locate other players. Those players must select hiding places which are difficult to discover while providing access for them to run safely to home base when the way is clear. The goal of this is to develop simulated agents that can play hide and seek (or more dangerous games). Due to the competitive nature of the game, reasoning must take place quickly in dynamically changing hostile surroundings.

This paper presents part of ZAROFF[1] a system for generating an *animated simulation* of humans playing hide and seek. (Figure 1 shows frames from one game.) We describe a planning system for the seeker and its vertical integration into a system that selects reactive behaviors to execute in an animated simulation. Operation of the planner is interleaved with execution of the reactive behaviors so that the agent may adapt to a dynamic environment. Adaptivity is also supported through least-commitment planning, as the planner only looks ahead one action at each level of its abstraction hierarchy.

The planner described in this system has been ported from an initial domain that combined search and manipulation tasks (Geib, Levison, & Moore 1994) to this new domain which requires locomotion. The software chosen for this work is *Jack*® (Badler, Phillips, & Webber 1993) running on Silicon Graphics workstations. *Jack* is a human modeling and simulation program developed at the Center for Human Modeling and Simulation at the University of Pennsylvania, that features visually realistic human locomotion based on both kinematic and dynamic techniques (Ko 1994). *Jack*'s LISP application programming interface (Becket 1994) is used to implement ZAROFF. This interface supports access to the environment (a database) and its behavioral simulation system.

In ZAROFF, a planner interacts with an animated simulation that provides a dynamically changing environment to which the planner has only limited perceptual access. These environmental characteristics motivate our system architecture.

Interacting with the environment

Our planner interacts with a multi-agent environment that consists of a database of graphical entities (e.g. geometric objects, human figures) and a behavioral simulation engine that moves objects in the database. The database records data on three-dimensional geometric figures with position and orientation. The planner's access to database information is restricted, to better simulate the limits of human perception. The planner's actions either directly manipulate the database, or indirectly affect the database by influencing the behavioral simulation engine.

Perception in ZAROFF

By limiting the planner's access to the database, the planner only has access to information about objects which the seeker can "see". For the planner to decide on an action, it must see all the objects involved in that action.

We implement a model of perception which restricts access to only those objects in the environment which are in a direct line of sight from the seeker. This is approximated using the graphics technique of ray-casting.

*This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory; ARPA AASERT DAAH04-94-G-0362; DMSO DAAH04-94-G-0402; ARPA DAMD17-94-J-4486; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; DMSO through the University of Iowa; and NSF CISE CDA88-22719.

[1] Named for the hunter in Richard Connell's 1924 short story, *The Most Dangerous Game*

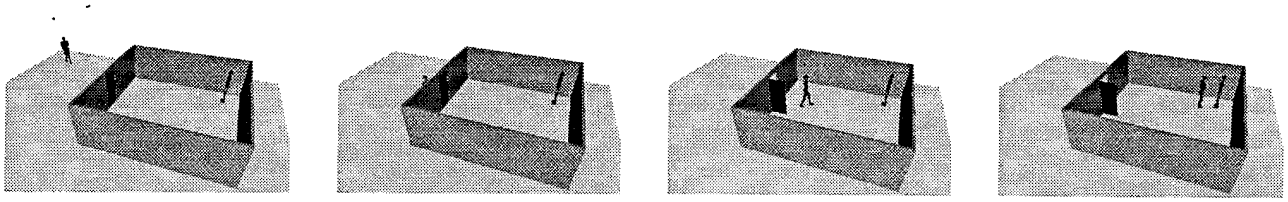


Figure 1: Frames from the conclusion of a game

Rays are cast from the seeker's eyes toward other figures. If any of these rays hit a figure before intersecting another object, that figure is perceivable. While "perception" is not synthetic vision, it satisfies the same role of forcing information-acquisition actions and motivates our use of a special purpose search planner.

This model of perception is less restrictive than that used in HOMER (Vere & Bickmore 1990), another animated simulation, which limited distance (only close objects can be seen) and angle of view (only objects in front can be seen) in a two-dimensional environment.

Action in ZAROFF

Actions chosen by the planner are carried out by an *action execution* module (see Figure 2). Both components are well matched to the dynamic environment in which ZAROFF acts: the planner quickly selects the next action to perform based on comparison between the perceived world state and an incomplete hierarchical plan that is regularly revised. The action execution module controls locomotion in a manner reactive to changes in the terrain and moving objects.

System Architecture

Our division of the control of a seeker between a *planning component* (the general purpose planner and special purpose search planner) and a *reactive behavior component* (the action execution module) reflects a distinction between deliberative and non-deliberative actions. Keeping track of where you are located in a complex environment and what hiding places have been checked requires deliberate effort, while walking from one place to another generally does not. Together, these two components create realistic animations of human decision-making and locomotion while playing hide and seek.

Figure 2 depicts information flow in ZAROFF. The system starts by initializing the plan with the input goal (finding a hiding human), populating the database with the initial locations of all the objects and human figures in the simulation, and creating a partial map from what the seeker can see around him. The planner and the Behavioral Simulation System start processing simultaneously. The planner queries the state of the database through the Filtered Perception module to decide how to elaborate the plan and select an action. If necessary, the Search Planner is consulted to assist in planning how to find things. When the planner decides on an action, it instructs Action Execution to carry it out. Further planning is suspended until the action has terminated (successfully or unsuccessfully).

In making decisions about what to do next, each component makes use of its own internal simulation, which differs from the graphical animation of the environment. The planner uses abstract descriptions of the effects of each action to choose one which will move closer to the specified goal. The search planner simulates the movements of an agent on its internal map of the environment. Action Execution simulates taking the next step in several alternate locations. At each level of decision making, an internal simulation is used *at an appropriate granularity.*

Action Execution

The Action Execution module is responsible for the control of all actions occurring in ZAROFF. Most actions such as opening and closing doors are performed directly by this module. Human locomotion is a special case which is performed by the Behavioral Simulation System (BSS) (Becket & Badler 1993; Badler, Phillips, & Webber 1993). Action Execution controls this locomotion indirectly.

Non-locomotion actions are performed directly by Action Execution manipulating the environment. For example, a door is opened by rotating it about its hinges. This rotation is done incrementally, a small amount each frame of animation.

Locomotion is performed indirectly by Action Execution creating *sensors* and binding them to human figures in the database. Since BSS is constantly monitoring the environment, this immediately initiates the appropriate agent locomotion.

Neither path-planning nor explicit instructions are used to drive locomotion; agent control and apparent complexity are the result of the interaction of a few relatively simple behaviors with a complex (and changing) environment. An agent is made aware of its environment through the use of a network of sensors. Based on the information gathered by these sensors the path through the terrain is incrementally computed. This allows the agent to react to unexpected events such as moving obstacles, changing terrain, or a moving goal (Reich *et al.* 1994).

Sensors

A sensor is a function which maps a human's position and orientation (his state) in the environment to

a stress value, where lower values represent more desirable states. The agent (here, the seeker) utilizes a set of sensors in interacting with its environment. Currently there are four classes of sensors:

Attraction: An attraction sensor (attractor) is used to draw the seeker toward a goal, either an object or a location. If a goal object moves, the point of attraction moves appropriately. The sensor output (stress value) of an attractor is high when the agent is far from the goal and decreases as the agent nears the goal.

Repulsion: A repulsion sensor (repulser) is used to avoid collisions between the seeker and objects. Repulsers have a sector-shaped region of sensitivity. If there are no objects in this region the sensor output is zero. Otherwise the output is proportional to the distance and size of the detected objects.

Field-of-View: A field-of-view sensor determines whether or not the agent is visible to any other agent. (It will be used to support the players who must hide.) The sensor output is proportional to the number of agents' fields-of-view it is in, and inversely proportional to the distances to these agents.

The Behavioral Simulation System

BSS provides general locomotion of objects in *Jack*, and is used in ZAROFF to generate human locomotion. The central control mechanism of BSS is a loop that includes perception, control, and action. During the perception phase the sensors are polled, during the control phase the next foot position is selected, and during the action phase the step is taken.

General purpose planning

The next two sections describe the system for planning the overall behavior of the seeker agent, which combines a hierarchical planner with a special purpose search planner.

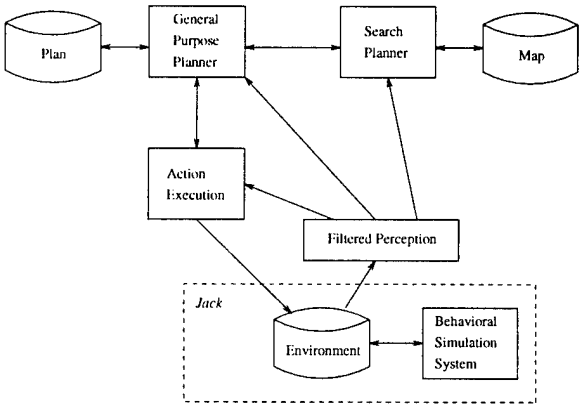


Figure 2: Information flow in ZAROFF

ITPLANS (Geib 1992) is a hierarchical planner, in which hierarchical expansion only takes place to the degree necessary to determine the next action to be carried out. It consists in an incremental expansion of the frontier of the plan structure to successively lower levels of abstraction. The incremental nature of the plan allows the system to make commitments at the appropriate level of detail for action while not committing the system to future actions that might be obviated by changes in the world. The close coupling of ITPLANS with the environment manifests itself in two ways:

First, the traversal and pruning process the planner follows at each interval relies on being able to determine the actual state of the world and compare that with its goals. During the expansion process ITPLANS examines the state of the world and its memory to determine if any of the goals within its plan have been satisfied. When a goal has been satisfied serendipitously, it can be pruned out of the plan structure, and the system can move on to consider its next goal.

Second, ITPLANS "leans on the world" (Agre 1988) when predicting the results of its actions. Rather than maintaining a complete model of the world and the state that results from executing the action, ITPLANS uses a simpler method based on associating conditional add and delete lists with each action. ITPLANS assumes that a given proposition is true in the state that results from the action if (1) the proposition is explicitly added by the add list or (2) the proposition is true *now* in the world and it is not contained on the delete list. By this method, ITPLANS can make predictions about the results of executing an action without modeling the entire world state.

Search planning

A consequence of limited perception is the occasional need to find objects. Our approach is to isolate this reasoning in a specialized module, a *search planner* that translates information acquisition goals to high-level physical goals to explore parts of the environment. As Haas points out (Haas 1993), any plan for acquiring information must rest on what the agent *knows* about the environment. That is, in order to search for an object, an agent must know (or discover during the search) the regions of space where the object might be.

Searches terminate successfully when a referent object is seen in the environment. They terminate unsuccessfully when there are no more regions to explore. A search may also be terminated if the environment changes in a way that obviates the search.

Maintaining a Map

Our approach to search planning relies on maintaining information about the state of a heuristic search on an internal map. The heuristic search has finding a desired object as its goal. It uses distance from the agent to order regions for exploration. Two lists of regions

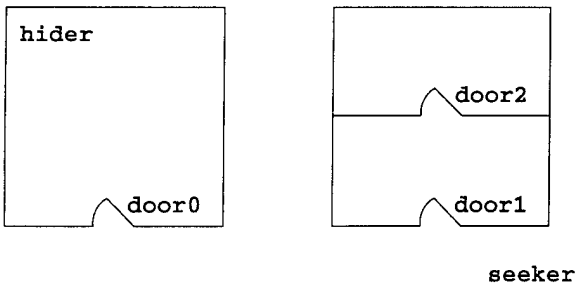


Figure 4: Plan view of example environment

are maintained by the search algorithm, an *open* list of regions yet to be explored and a *closed* list of regions which have been explored. Its internal map consists of nodes that correspond to bounded regions connected by links that correspond to doors.

In one iteration of the search, the closest region on the open list is selected to be explored. ITPLANS generates a plan for going to and exploring that region, opening any doors necessary along the way. After each action in this plan is executed, the resulting world is observed to determine if the desired object has been located. New doors and regions observed during the action are added to both the map and the open list.

Pemberton and Korf (Pemberton & Korf 1992) present optimal algorithms for heuristic search on graph spaces, where only a portion of the graph is available before the agent must commit to an action. We use their Incremental Best-First Search (IBFS) algorithm, which uses best-first search to find the closest known open node. Heuristic estimates for this known part of the graph are recalculated as necessary.

Example

Having given an overview of the system's components, we now illustrate ZAROFF with an example drawn from a game of hide and seek. To illustrate the conduct of a search, we will use an example environment with two buildings, one of which has two internal rooms separated by a door (Figure 4). Consider the seeker's goal of finding a hiding player. This is specified as `goto(X)` with the added constraint that `type(X) = HUMAN`.

ITPLANS considers the action `goto(X)` to be primitive but underspecified since the variable `X` is not bound to a particular object of type `HUMAN`. In order to bind the variable, the search planner must be called to generate a plan for locating a `HUMAN`. To this end, ITPLANS adds to the plan a *find node* and calls the search planner to instantiate a search plan (Figure 3a).

The search planner reasons from this knowledge acquisition goal of locating a `HUMAN`, to the goal of exploring regions where a `HUMAN` might be. Satisfying this goal requires physically searching through possible regions.

ITPLANS asks the search planner to expand the find node. Each time a find node is expanded, the search planner first examines the *Jack* environment to determine if an object of the specified type is visible to the agent. If not, the search planner selects a region to explore next, generates a goal to explore that region, and adds it to the plan (Figure 3b). The initial map (Figure 5) of regions has two doors for the search planner to choose from; the closest, `door1`, is recommended for exploration. This goal is then further expanded by ITPLANS to go to `door1` and open it (Figure 3c). Since all of the arguments in the first action are bound to specific objects, it can be carried out. Action Execution performs this action indirectly by binding an attraction sensor to the seeker. When the seeker arrives, `door1` is opened directly by Action Execution.

After `door1` is opened, ITPLANS uses the search planner to evaluate the progress of the search by examining the world for objects having the property `HUMAN`. If one is located, the search is considered successful. If not, the search planner selects a new region for exploration and the searching process repeats until there are no more regions to explore.

In this case, opening `door1` does not reveal a `HUMAN`, but does permit the agent to see another door, `door2`, that is automatically added to the search planner's internal map. As `door2` is the closest unexplored space, on the next iteration the planner will plan to explore behind `door2`. Opening `door2` does not reveal the `HUMAN`, so the search proceeds to the next closest unexplored region – the other building.

Here we see the advantage of maintaining a map. Immediately after opening `door2`, the agent is inside one building and decides to go to the door of the other building. Since this destination region is known (from having seen it previously), we could simply take the action `goto(door0)`. This would result in the agent walking directly toward the door until stopped by the wall. To avoid getting caught in this local minimum, the search planner uses its internal map (Figure 6) to plan a path to the next region. The only known path to `door0` is to exit the current building through `door1`. The search planner returns this sequence of intentions

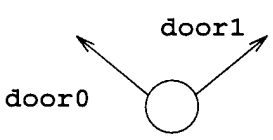


Figure 5: Initial map

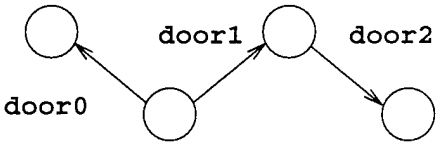


Figure 6: Final map

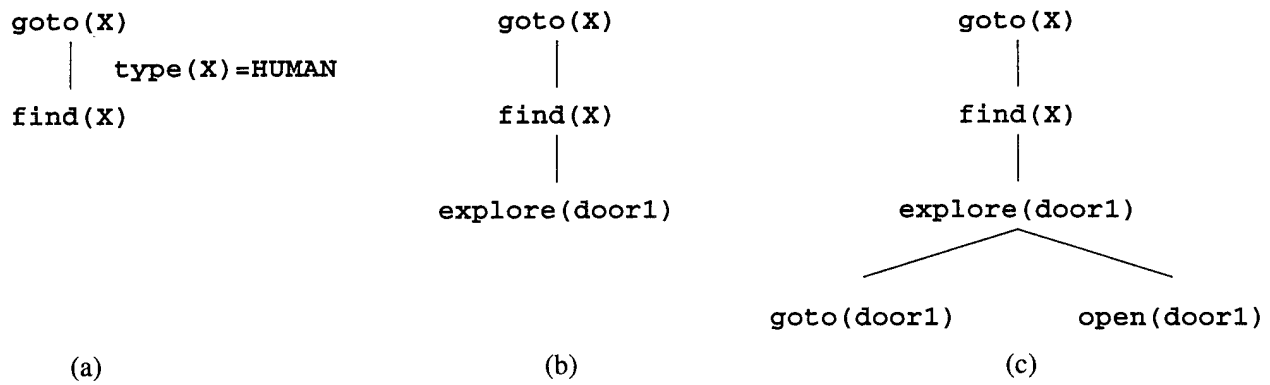


Figure 3: Evolution of the plan graph

to ITPLANS, which then invokes Action Execution to generate locomotion along this path. After opening `door0`, the seeker finally sees a `HUMAN` and can go to it.

Conclusion

We have implemented a plan-based controller for the *seeker* role in the game of hide and seek. Our agent dynamically reacts to changes in the environment, from the level of terrain up to changes in information about where the other players may be hiding. The implementation combines general purpose planning, special purpose reasoning about conducting a search, and reactive control of human behaviors.

ZAROFF is an effective system for animating humans carrying out tasks that require locomotion. Limiting the human agent's awareness of its environment by simulated perception increases the realism of the behavior generated.

Acknowledgements

We wish to thank our advisors Norm Badler and Bonnie Webber for their support of this work. Thanks also to them, Welton Becket, Jonathan Crabtree, Brett Douville, and Jeff Nimeroff for commenting on drafts of this paper.

References

Agre, P. 1988. The dynamic structure of everyday life. Technical Report 1085, MIT Artificial Intelligence Laboratory.

Badler, N.; Phillips, C.; and Webber, B. 1993. *Simulating Humans: Computer Graphics, Animation and Control.* Oxford University Press.

Becket, W., and Badler, N. I. 1993. Integrated behavioral agent architecture. In *Proceedings of the Third Conference on Computer Generated Forces and Behavior Representation*, 57–68.

Becket, W. M. 1994. The Jack LISP API. Technical Report MS-CIS-94-01, University of Pennsylvania, Philadelphia, PA.

Geib, C. W.; Levison, L.; and Moore, M. B. 1994. SodaJack: An architecture for agents that search for and manipulate objects. Technical Report MS-CIS-94-16/LINC LAB 265, Department of Computer and Information Science, University of Pennsylvania.

Geib, C. 1992. Intentions in means-end planning. Technical Report MS-CIS-92-73, Department of Computer and Information Science, University of Pennsylvania.

Haas, A. 1993. Natural language and robot planning. Technical Report 9318, Department of Computer Science, SUNY Albany.

Ko, H. 1994. *Kinematic and Dynamic Techniques for Analyzing, Predicting, and Animating Human Locomotion.* Ph.D. Dissertation, University of Pennsylvania.

Pemberton, J. C., and Korf, R. E. 1992. Incremental path planning on graphs with cycles. In *Proceedings of AIPS 92*, 179–188.

Reich, B. D.; Ko, H.; Becket, W.; and Badler, N. I. 1994. Terrain reasoning for human locomotion. In *Proceedings of Computer Animation '94*, 996–1005. Geneva, Switzerland: IEEE Computer Society Press.

Vere, S., and Bickmore, T. 1990. A basic agent. *Computational Intelligence* 6:41–60.

H Production and Playback of Human Figure Motion for 3D Virtual Environments: Granieri, Crabtree, and Badler

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

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

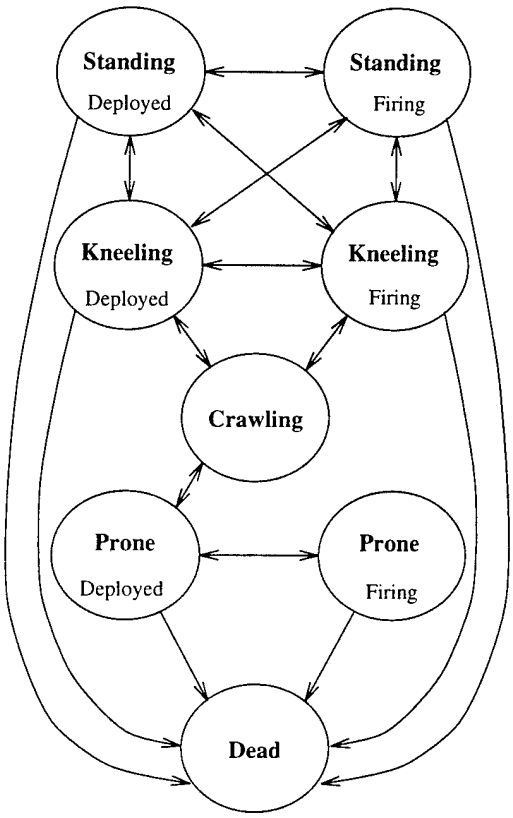


Figure 2: The *static* posture graph

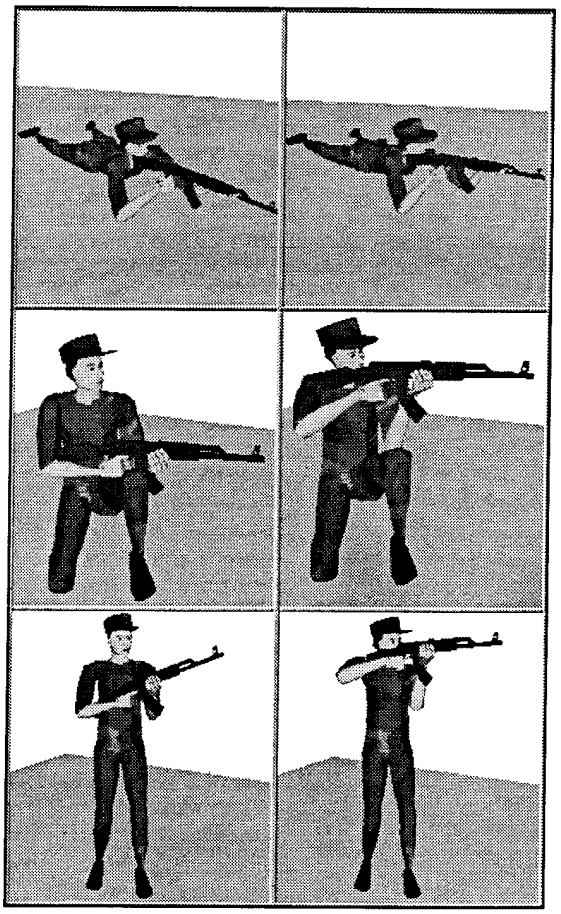


Figure 3: Some of the static postures a soldier can take in DIS

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

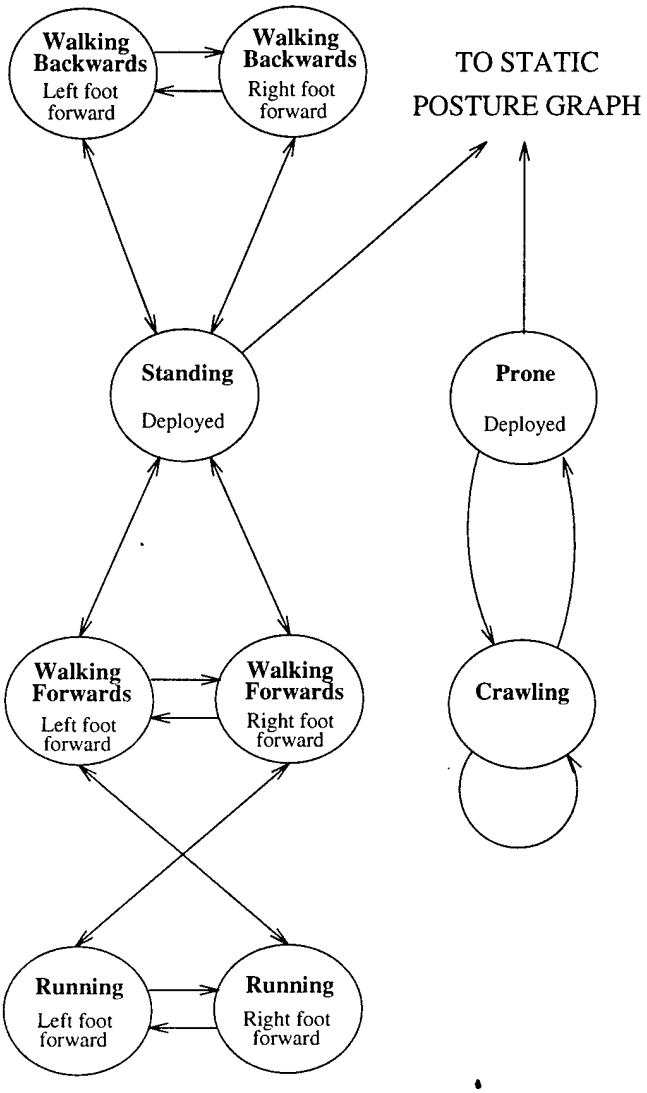


Figure 4: The *locomotion* posture graph

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

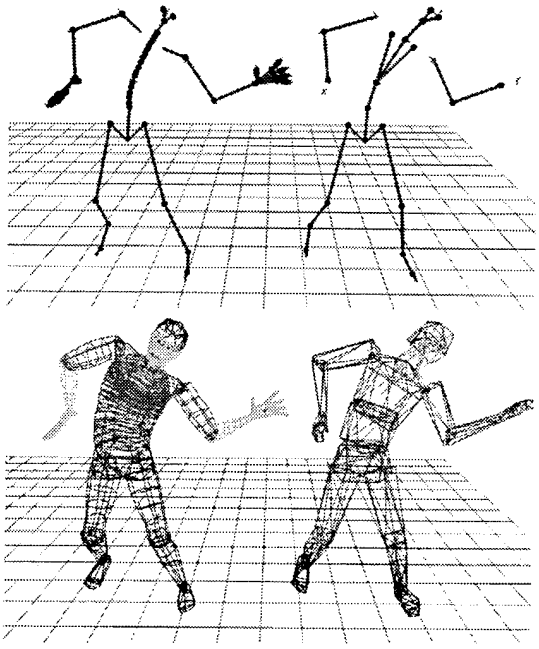


Figure 6: human–1 and human–2 models during slaving. human–1 is the master. Upper window is the skeletal articulation. Models are offset for illustrative purposes.

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

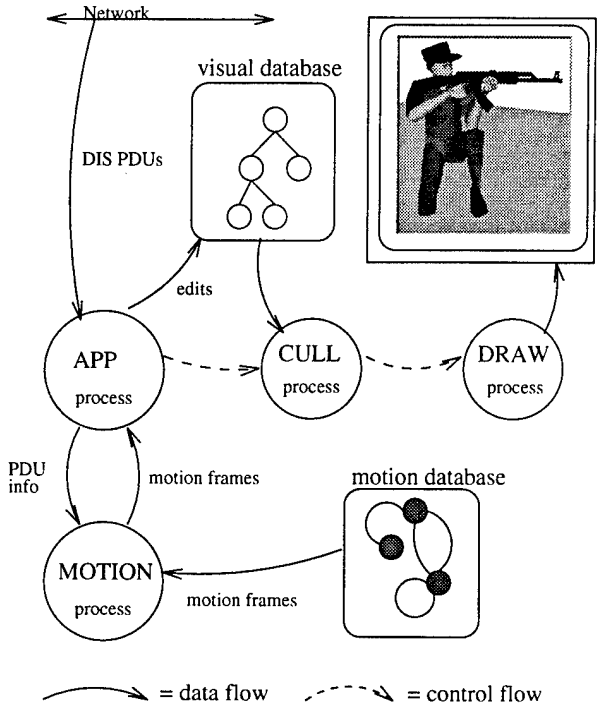


Figure 7: Overview of multi-processing framework for run-time system.

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 runtime 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 nonzero 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 $0, 3, 6, ..., 120$. 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.

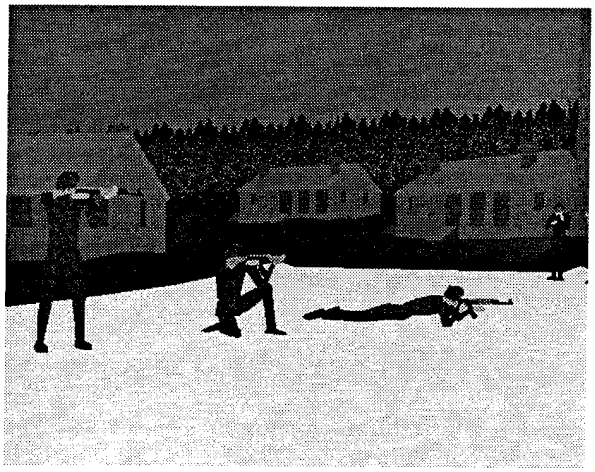


Figure 8: A View of Battle Town with several soldiers in different postures

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.

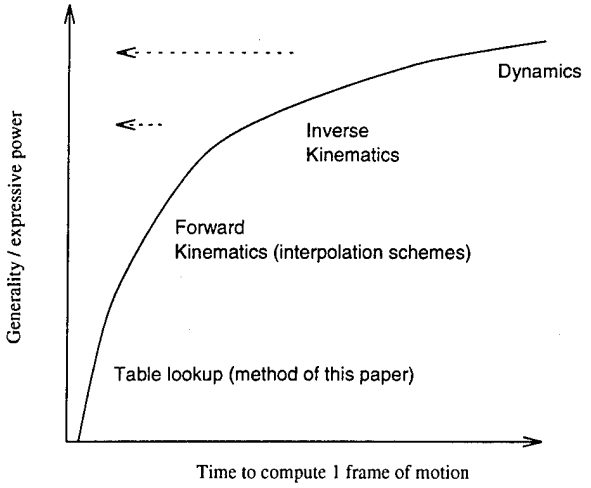


Figure 9: Trade-off between time and generality for motion generation techniques

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.