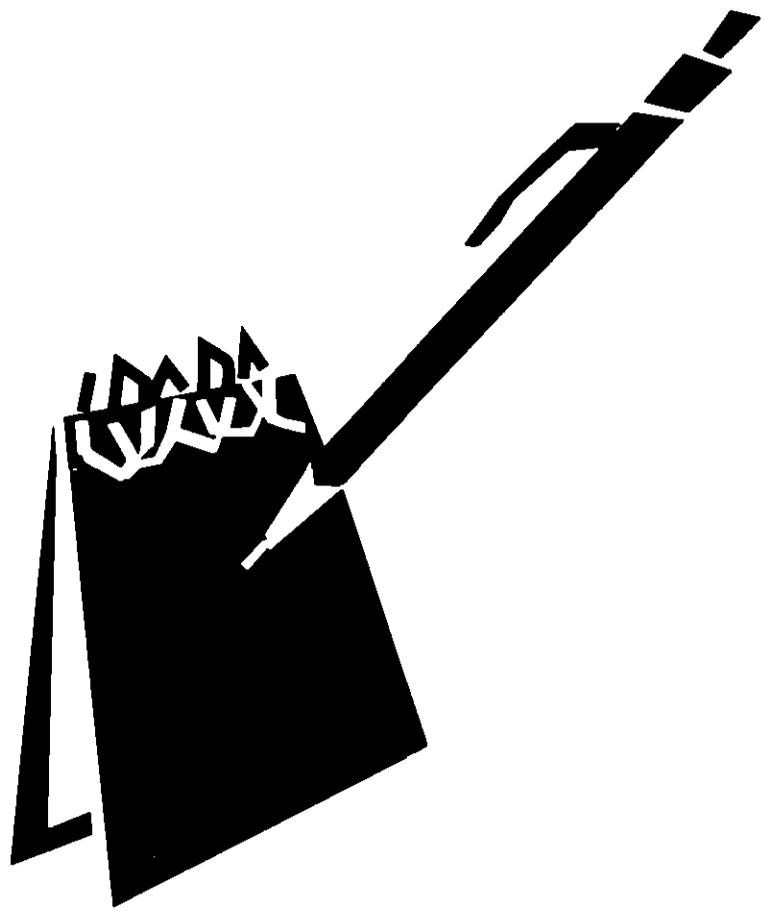


89
T
A
N
C
G
I
S

28

Virtual Humans: Behaviors and Physics, Acting and Reacting



25th International Conference on Computer Graphics and Interactive Techniques
Exhibition 21-23 July 1998 Conference 19-24 July 1998
Orlando, Florida USA

course notes



28

Virtual Humans Behaviors and Physics, Acting and Reacting

Organizer

Norman Badler

University of Pennsylvania

Lecturers

Norman Badler

Dimitris Metaxas

University of Pennsylvania

Armin Bruderlin

Sony Pictures Imageworks

Athomas Goldberg

Ken Perlin

New York University

Nadia Magnenat Thalmann

University of Geneva

25th International Conference on Computer Graphics and Interactive Techniques

Exhibition **21 23 July 1998** Conference **19 24 July 1998**

Orlando, Florida USA

course notes

Real-Time Virtual Humans

Norman Badler

Armin Bruderlin

Ken Perlin

Athomas Goldberg

Nadia Magnenat-Thalmann

Dimitris Metaxas

July 21, 1998

SIGGRAPH '98 Course 28

1

Schedule

8:30 - 9:45 Badler

9:45 - 10:00; 10:15 - 11:00 Bruderlin

11:00 - 12:00; 1:30 - 1:45 Perlin/Goldberg

1:45 - 3:00 Magnenat-Thalmann

3:15 - 4:30 Metaxas

4:30 - 5:00 Panel (all)

July 21, 1998

SIGGRAPH '98 Course 28

2

Virtual Humans

Norman I. Badler

***Center for Human Modeling and
Simulation***

University of Pennsylvania

Philadelphia, PA 19104-6389

215-898-5862 phone; 215-573-7453 fax

<http://www.cis.upenn.edu/~badler>

July 21, 1998

SIGGRAPH'98 Course 28

3

What are Virtual Humans?

***Computer models of people that can be used
as substitutes for "the real thing" in***

- Evaluating ergonomics prior to
actual construction of some system.***
- Representing ourselves or other live
or constructed participants in virtual
environments***

July 21, 1998

SIGGRAPH'98 Course 28

4

Applications for Virtual Humans:

- *Engineering Ergonomics.*
- *Maintenance Assessment.*
- *Games/Special Effects.*
- *Military Simulations.*
- *Job Education/Training.*
- *Medical Simulations.*

July 21, 1998

SIGGRAPH'98 Course 28

5

Virtual Human "Dimensions"

- *Appearance*
- *Function*
- *Time*
- *Autonomy*
- *Individuality*

July 21, 1998

SIGGRAPH'98 Course 28

6

Appearance:

*2D drawings > 3D wireframe >
3D polyhedra > curved surfaces >
freeform deformations >
accurate surfaces > muscles, fat >
biomechanics > clothing, equipment >
physiological effects (perspiration,
irritation, injury)*

July 21, 1998

SIGGRAPH '98 Course 28

7

Function:

*cartoon > jointed skeleton >
joint limits > strength limits >
fatigue > hazards > injury > skills >
effects of loads and stressors >
psychological models >
cognitive models > roles > teaming*

July 21, 1998

SIGGRAPH '98 Course 28

8

Time (~Number):

off-line animation >
interactive manipulation >
real-time motion playback >
parameterized motion synthesis >
multiple agents >
crowds > coordinated teams

July 21, 1998

SIGGRAPH'98 Course 28

9

Autonomy:

drawing > scripting >
interacting > reacting >
making decisions >
communicating > intending >
taking initiative > leading

July 21, 1998

SIGGRAPH'98 Course 28

10

Individuality:

generic character >

hand-crafted character >

cultural distinctions > personality >

psychological-physiological profiles >

gender and age >

specific individual

July 21, 1998

SIGGRAPH '98 Course 28

11

Comparative Virtual Humans

<i>Application</i>	<i>Appear.</i>	<i>Function</i>	<i>Time</i>	<i>Autonomy</i>	<i>Individ.</i>
Cartoons	<i>high</i>	<i>low</i>	<i>high</i>	<i>low</i>	<i>high</i>
Games	<i>high</i>	<i>low</i>	<i>low</i>	<i>med</i>	<i>med</i>
Sp. Effects	<i>high</i>	<i>low</i>	<i>high</i>	<i>low</i>	<i>med</i>
Medical	<i>high</i>	<i>high</i>	<i>med</i>	<i>med</i>	<i>med</i>
Ergonomics	<i>med</i>	<i>high</i>	<i>med</i>	<i>med</i>	<i>low</i>
Education	<i>med</i>	<i>low</i>	<i>low</i>	<i>med</i>	<i>med</i>
Tutoring	<i>med</i>	<i>low</i>	<i>med</i>	<i>high</i>	<i>low</i>
Military	<i>med</i>	<i>med</i>	<i>low</i>	<i>med</i>	<i>low</i>

July 21, 1998

SIGGRAPH '98 Course 28

12

What Makes a Virtual Human Human?



- *Degree of human-like appearance is relative to application.*
- *Apparent decision-making and emotional reactions yield believability.*
- *Communicating intentions builds community.*

July 21, 1998

SIGGRAPH '98 Course 21

13

Control for Interactivity



- *Now mostly point-and-click.*
 - *Need language-like commands (text or speech).*
- Move the human interface toward a command or instructional view -- as if the Virtual Human were another real person.**

July 21, 1998

SIGGRAPH '98 Course 21

14

Control for Autonomy

- *Provide human-like reactions and decision-making ("AI").*
- *Personality, roles, culture, skills, perceptions, and intelligence affect interaction with the environment, situation, and other agents.*

July 21, 1998

SIGGRAPH '98 Course 28

15

Human-Like Appearance

Require human-like structure

- Spine and neck
- Shoulder, clavicle, and arm

Require human-like surfaces

- Smooth skin
- Face
- Synthetic clothing

July 21, 1998

SIGGRAPH '98 Course 28

16

Basic Human Movement Capabilities

- *Gesture / Reach / Grasp.*
- *Walk / Orient / Locomote.*
- *Visual Attention / Search.*
- *Pull / Lift / Carry.*
- *Motion playback (previously scripted or stored, e.g. parameterized motion capture).*

July 21, 1998

SIGGRAPH '98 Course 28

17

Arms and Hands

- *Object-specific reasoning for approach, grasping, and use.*
- *Gesture artifacts during communication.*
- *Fast position and orientation inverse kinematics.*
- *Two-handed grasps.*

July 21, 1998

SIGGRAPH '98 Course 28

18

Synthesized Motions

- *Inverse kinematics for arms, legs, spine.*
- *Paths or footsteps driving locomotion.*
- *Balance constraint on whole body.*
- *Dynamics control from forces and torques.*
- *Facial expressions*
- *Secondary motions to enhance simpler forms (feet, blinks, eye gaze, breathing).*

July 21, 1998

SIGGRAPH '98 Course 28

19

Locomotion

- *Agent given attributes such as motivation, directedness, speed.*
- *Sensors available: attractor, repulser, range, terrain, low obstacle, humans, etc.*
- *Anticipation (prediction).*
- *Chasing, hiding, evading, searching, climbing, etc.*

July 21, 1998

SIGGRAPH '98 Course 28

20

Attention

- *Is not usually stated explicitly.*
- *Is a resource.*
- *Monitors a task queue.*
- *Attention and other sensing actions consume time.*
- *Produces proactive behaviors.*

July 21, 1998

SIGGRAPH '98 Course 28

21

Parallel Transition Networks (PaT-Nets)

A Virtual Parallel Execution Engine for virtual human actions:

- *Processes are nodes.*
- *Instantaneous (conditional or probabilistic) transitions are edges.*
- *Message passing and synchronization.*
- *Lisp and C++ versions.*

July 21, 1998

SIGGRAPH '98 Course 28

22

PaT-Net Applications

- *Gesture, head motion, and eye gaze during conversation. (SIGGRAPH 94)*
- *Hide and seek. (VRAIS '96)*
- *Physiological state under trauma and treatment. (Presence J. 96)*
- *Jack Presenter. (AAAI-97 Workshop)*
- *JackMOO. (WebSim '98)*

July 21, 1998

SIGGRAPH '98 Course 28

23

"Smart Avatars" (3rd person VR)

- *Provide REAL-TIME human-like appearance, reactions, and decision-making.*
- *Ultimately personality, roles, culture, skills, perceptions, and intelligence should affect interaction with the environment, situation, and other agents.*

July 21, 1998

SIGGRAPH '98 Course 28

24

JackMOO Components

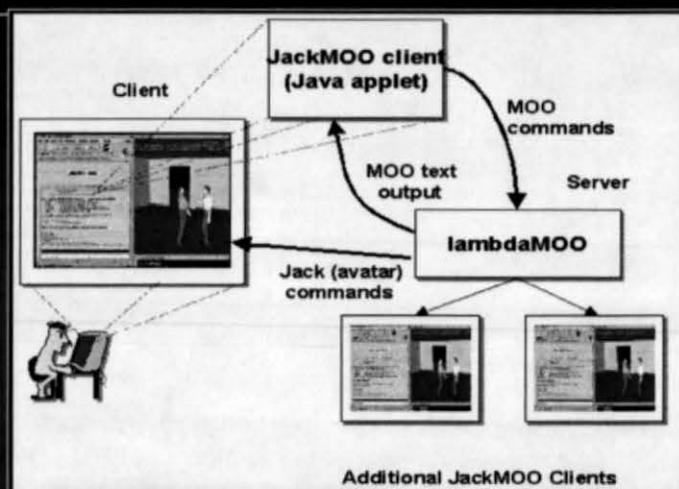
- **Jack**: virtual human creation and motion authoring system
- **lambdaMOO**: multi-user, network accessible, programmable, interactive system from Xerox Parc
- **Client**: Java applet providing user interface and control flow mediation

July 21, 1998

SIGGRAPH '98 Course 28

25

Architecture



July 21, 1998

SIGGRAPH '98 Course 28

26

Experiment 1

Handshaking Scenario:

- Two avatars "TJ" and "Norm" get each other's attention; shake hands.

July 21, 1998

SIGGRAPH'98 Course 28

27

Experiment 2

Agent-Object-Preposition Interaction:

- Agent walks to and sits in chair, wherever placed; agent walks around chair.

July 21, 1998

SIGGRAPH'98 Course 28

28

Experiment 3

Relationship Scenario:

- One follows the other out of the room.

July 21, 1998

SIGGRAPH '98 Course 28

29

Observations

- lambdaMOO verbs (simple imperative sentences) linked to multi-user, distributed Jack capability.
- Jack PaT-Net programs provide API for lambdaMOO verbs.
- Need a representation for agent actions more compatible with language.

July 21, 1998

SIGGRAPH '98 Course 28

30

Connecting Language and Animation



Design a representation which is able to bridge concepts from both language and animation.

Implement semantics for human actions to create SMART AVATARS.

July 21, 1998

SIGGRAPH'98 Course 28

31

Parameterized Action Representation (PAR)



•Representation derived from BOTH NL analyses and animation requirements:

- *Agent, Objects, Sub-Actions*
- *Preconditions, Postconditions*
- *Applicability and Culmination conditions.*
- *Spatio-temporal terms.*
- *Agent Manner parameters.*

July 21, 1998

SIGGRAPH'98 Course 28

32

Software Architecture

NL Commands/NL Generation

PAR and database/lexicon

Execution Engine and global clock

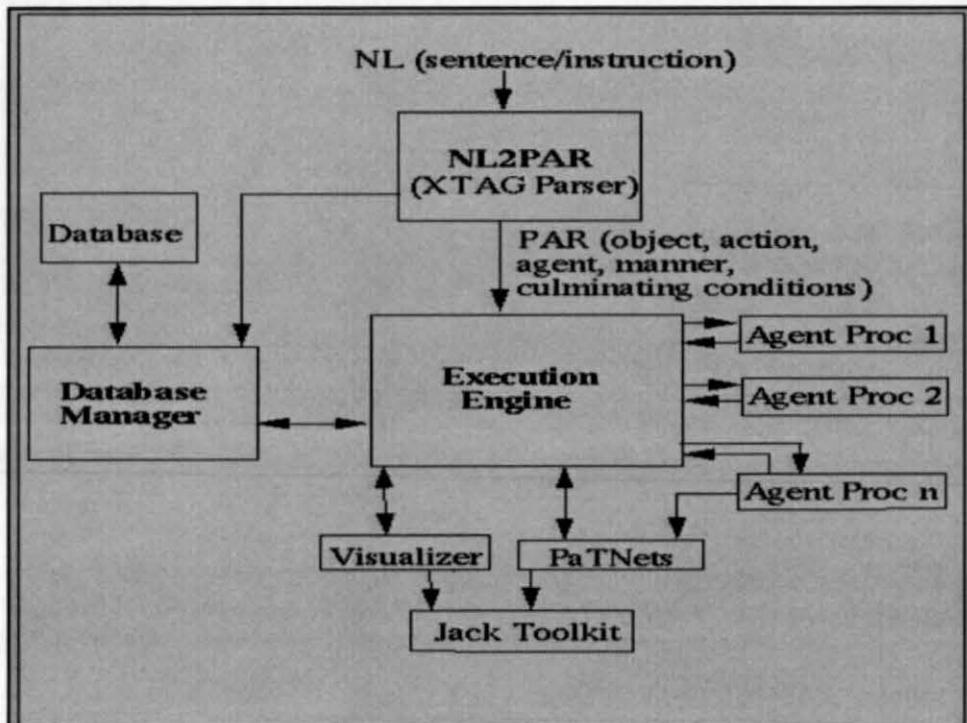
Agent Procedures and Motion Synthesis

OpenGL and Transom Jack Toolkit

July 21, 1998

SIGGRAPH '98 Course 28

33



Role of Linguistics in Knowledge Modeling (1)

Linguistic classifications based on distributional analysis can provide:

Typical properties of animate agents

Typical modifiers of animate agents

- Dimensions along which agent behavior can vary
- AGENT, +DELIBERATE, +CAREFUL, +STRONG (e.g.)

July 21, 1998

SIGGRAPH'98 Course 28

35

Role of Linguistics in Knowledge Modeling (2)

Typical actions of animate agents

Typical modifiers for actions

- Dimensions along which action execution can vary
- ACTION CAN BE PERFORMED CAUSALLY, CAREFULLY, GENTLY, etc.
- Agent manner executed via "Effort" basis functions.

Basing an agent and action ontology on linguistic evidence ensures extensibility

July 21, 1998

SIGGRAPH'98 Course 28

36

Work in Progress

- *PAR development and implementation.*
- *Various human model improvements (walking, motion models).*
- *Smart avatars.*
- *Multi-user virtual environments.*
- *Agent models with culture, roles, manner, and personality.*

July 21, 1998

SIGGRAPH '98 Course 28

37

Conclusions

- *Reactive, proactive, and decision-making behaviors needed for action execution.*
- *Individuals vary in perceptions of context.*
- *Language interfaces (through PAR) will improve access and usability.*
- *Hardware improvements alone will not yield intelligence.*

July 21, 1998

SIGGRAPH '98 Course 28

38

Virtual Humans

Norman Badler

Center for Human Modeling and Simulation
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389
badler@central.cis.upenn.edu

Abstract

The last few years have seen great maturation in the computation speed and control methods needed to portray 3D virtual humans suitable for real interactive applications. We first describe the state of the art, then focus on the particular approach taken at the University of Pennsylvania with the *Jack* system. Various aspects of real-time virtual humans are considered, such as appearance and motion, interactive control, autonomous action, gesture, attention, locomotion, and multiple individuals. The underlying architecture consists of a sense-control-act structure that permits reactive behaviors to be locally adaptive to the environment, and a **PaT-Net** parallel finite-state machine controller that can be used to drive virtual humans through complex tasks. We then argue for a deep connection between language and animation and describe current efforts in linking them through the *JackMOO* extension to lambdaMOO and an initial draft of a *Parameterized Action Representation*.

1 Virtual Humans

Only fifty years ago, computers were barely able to compute useful mathematical functions. Twenty-five years ago, enthusiastic computer researchers were predicting that all sorts of human tasks from game-playing to automatic robots that travel and communicate with us would be in our future. Today's truth lies somewhere in-between. We have balanced our expectations of complete machine autonomy with a more rational view that machines should assist people to accomplish meaningful, difficult, and often enormously complex tasks. When those tasks involve human interaction with the physical world, computational representations of the human body can be used to escape the constraints of presence, safety, and even physicality.

Virtual humans are computer models of people that can be used

- as substitutes for “the real thing” in *ergonomic* evaluations of computer-based designs for vehicles, work areas, machine tools, assembly lines, etc., *prior to the actual construction of those spaces*;
- for *embedding real-time representations of ourselves or other live participants* into virtual environments.

Recent improvements in computation speed and control methods have allowed the portrayal of 3D humans suitable for interactive and real-time applications. There are many reasons to design specialized human models that individually optimize character, performance, intelligence, and so on. Many research and development efforts concentrate on one or two of these criteria.

In the efforts that we describe here, we cross several domains which in turn build from various interrelated facets of human beings:

- **Human Factors Analysis:** Human size, capabilities, behavior, and performance affects work in and use of designed environments.
- **Real-Time Agents and Avatars:** People come from different cultures and have different personalities; this richness and diversity must be reflected in virtual humans since it influences appearance as well as reaction and choice.
- **Instruction Understanding and Generation:** Humans communicate with one another within a rich context of shared language, senses, and experience and this needs to be extended to computer-generated agents and avatars.
- **Bio-Medical Simulation:** The human machine is a complex of physical structures and functions; to understand human behavior, physiological responses, and injuries we need to represent biological systems.
- **Motion and Shape Analysis:** Understanding what we perceive when we see or sense the world leads to models of the physical world (physics) and the geometric shapes and deformations of objects.

From these virtual humans research areas, many current, emergent, or future major applications are enabled:

- **Engineering:** Analysis and simulation for virtual prototyping and simulation-based design.
- **Virtual-Conferencing:** Efficient tele-conferencing using virtual representations of participants to reduce transmission bandwidth requirements.

- **Interaction** Agents and avatars that insert real-time humans into virtual worlds with virtual reality
- **Monitoring** Acquiring, interpreting, and understanding shape and motion data on human movement, performance, activities, or intent
- **Virtual Environments** Living and working in a virtual place for visualization, analysis, training, or just the experience
- **Games** Real-time characters with actions and personality for fun and profit
- **Training** Skill development, team coordination, and decision making
- **Education** Distance mentoring, interactive assistance, and personalized instruction
- **Military** Battlefield simulation with individual participants, team training, and peace-keeping operations
- **Design/Maintenance** Design for access, ease of repair, safety, tool clearance, visibility, and hazard avoidance

Besides general industry driven improvements in the underlying computer and graphical display technologies themselves, virtual humans will enable quantum leaps in applications requiring personal and live participation

In building models of virtual humans, there are varying notions of *virtual fidelity*. Understandably, these are application dependent. For example, fidelity to human size, capabilities, and joint and strength limits are essential to some applications such as design evaluation, whereas in games, training, and military simulations, temporal fidelity (real-time behavior) is essential. Understanding that different applications require different sorts of virtual fidelity leads to the question of what makes a virtual human “right”?

- What do you want to do with it?
- What do you want it to look like?
- What characteristics are important to success of the application?

Unfortunately the state of research in virtual humans is not as advanced as to make the proper selection a matter of buying off the shelf systems. There are gradations of fidelity in the models: some models are very advanced in a narrow area but lack other desirable features.

In a very general way, we can characterize the state of virtual human modeling along at least five dimensions, each with a wide range of realizations. Some significant datapoints along each one are listed below.

- Appearance 2D drawings > 3D wireframe > 3D polyhedra > curved surfaces > freeform deformations > accurate surfaces > muscles fat > biomechanics > clothing equipment > physiological effects (perspiration irritation injury)
- Function cartoon > jointed skeleton > joint limits > strength limits > fatigue > hazards > injury > skills > effects of loads and stressors > psychological models > cognitive models > roles > teaming
- Time off-line animation > interactive manipulation > real time motion playback > parameterized motion synthesis > multiple agents > crowds > coordinated teams
- Autonomy drawing > scripting > interacting > reacting > making decisions > communicating > intending > taking initiative > leading
- Individuality generic character > hand-crafted character > cultural distinctions > personality > psychological physiological profiles > gender and age > specific individual

If we need to invoke them, the *appearance* of increasingly accurate physiologically and biomechanically-grounded human models may be obtained. We can create virtual humans with *functional* limitations that go beyond cartoons into instantiations of known human factors data. Animated virtual humans can be created in human *time* scales through motion capture or computer synthesis. Virtual humans are also beginning to exhibit the early stages of *autonomy* and intelligence as they react and make decisions in novel changing environments rather than being forced into fixed movements. Finally rather preliminary investigations are underway to create characters with *individuality* and personality who react to and interact with other real or virtual people [11, 12, 18, 39, 46, 52]

Across various applications different capabilities are required as shown in Table 1. A model that is tuned for one application may not be adequate for another. An interesting challenge to be build virtual human models with enough parameters to provide effective support cross several application areas.

The University of Pennsylvania has been very actively engaged in research and development of human-like simulated figures. Our interest in human simulation is not unique, but the complex of activities surrounding our approach is. The framework for our research is a software system called *Jack*[®] [6]. *Jack* is an interactive system for definition, manipulation, animation, and performance analysis of virtual human figures¹. Our philosophy has led to a particular realization of a virtual human model that pushes the above five dimensions toward the more complex features.

- can be substituted for live individuals for workspace or cockpit evaluation

¹ *Jack* is now the basis of a commercial product distributed by Transom Technologies, Inc., of Ann Arbor, MI

Application	Appearance	Function	Time	Autonomy	Individuality
Cartoons	high	low	high	low	high
Games	high	low	low	med	med
Sp Effects	high	low	high	low	med
Medical	high	high	med	med	med
Ergonomics	med	high	med	med	low
Education	med	low	low	med	med
Tutoring	med	low	med	high	low
Military	med	med	low	med	low

Table 1 Capability requirements of several Virtual Human Applications

- demonstrates various (useful) human limitations, constraints, and capabilities
- may be moved live (in real-time) by position and orientation information or other motion generators such as *walk-to* or *look-at*
- may have its actions synthesized by a program so that it can make its own decisions, navigate spaces, and so on
- represents “anyone” rather than a single specific person or character

Virtual humans are different than simplified cartoon and game characters. What are the characteristics of this difference and why are virtual humans more difficult to construct? After all, anyone who goes to the movies can see marvelous synthetic characters but they have been created typically for one scene or one movie and are not meant to be re-used (except possibly by the animator – and certainly not by the viewer). The difference lies in the *interactivity* and *autonomy* of virtual humans. What makes a virtual human *human* is not just a well-executed exterior design but movements, reactions, and decision-making which appear “natural,” appropriate, and contextually-sensitive. Communication by and with virtual humans gives them a uniquely human capability: they can let us know their intentions, goals, and feelings thus building a bridge of empathy and understanding. Ultimately we should be able to communicate with virtual humans just as if they were real.

2 Agents and Avatars

We will consider an *agent* to be a virtual human figure representation that is created and controlled by computer programs. An *avatar* is a virtual human controlled by a live participant². The principal issues roughly follow the dimensions cited above: *appearance and motion*, mechanisms of *control for interactivity and autonomy*, including gesture, attention, and locomotion, and multi-agent *interaction, cooperation and coordination*.

2.1 Appearance and Motion

Avatars can be portrayed visually as 2D icons, cartoons [36], composited video, 3D shapes, or full 3D bodies [5, 55, 50, 54]. We are mostly interested in portraying human-like motions, so naturally tend toward the more realistic surface and articulation structures. In general, we prefer to design motions for highly articulated models and then reduce both the model detail and the articulatory detail as demanded by the application [29].

Along the appearance dimension, the *Jack* figure has developed as a polygonal model with rigid segments and joint motions and limits accurate enough for ergonomics evaluations [6]. For real time avatar purposes, simpler geometry can be used provided that the overall impression is one of a task-relevant figure. Thus a soldier model with 110 polygons is acceptable if drawn small enough and colored and/or texture mapped to be recognized as a soldier. On the other hand, a vehicle occupant model must show accurate and visually continuous joint geometry under typical motions. It must be both an acceptable occupant surrogate as well as a pleasing model for the non-technical viewer – who may be used to going to the movies to see the expensive special effects figures. Our “smooth body” [3, 57] was developed using free-form deformation techniques [53] to aid in the portrayal of visually appealing virtual humans.

The motions manifest in the avatar may arise from various sources

- Motion capture from direct live video
- Motion capture from sensors
- Pre-stored motion data
 - as 2D sprites
 - as 3D global transformations
 - as 3D local (joint) transformations

²Interest in avatars was fueled by Neil Stephenson's novel *Snow Crash*. An analysis of the characteristics of the *Snow Crash* avatars shows that they are [individually] fundamentally achievable with today's technologies [2].

- Motion synthesis
 - joint angle interpolation
 - inverse kinematics
 - dynamics
 - other generators (e.g. locomotion, faces)

In general, we will not consider 2D or purely video presentations of avatars, rather we will concentrate on avatars that more-or-less mimic human structure

The distinction between “synthesized” motions and the other types is roughly that the former generate transformations for more than one joint at a time. Thus, for example, we store a time series of joint angle changes (per joint) in *channelsets* so that specific motions can be re-played under real-time constraints [29]. No deviation from the pre-stored local transformations are allowed, although the whole body may be re-oriented or the playback speed varied. In a particularly effective modification of this technique, Perlin adds periodic noise to real-time joint transformations to achieve greater movement variability, animacy, and motion transitions [45].

In a motion synthesizer, a small number of parameters control a much greater number of joints, for example

- end effector position and orientation can control joints along an articulated chain [61, 34, 58],
- a path or footsteps can control leg and foot rotations through a locomotion model [27, 33],
- a balance constraint can be superimposed on gross body motions [6, 33],
- dynamics calculations can move joints subject to arbitrary external and internal applied forces [35, 40],
- facial expressions can add a personal and human outlook [18, 44],
- secondary motions can enhance a simpler form (noise, blinks, gestures, eye gaze, foot motions) [18, 45, 30]

The relative merits of pre-stored and synthesized motions must be considered when implementing virtual humans. The advantages to pre-stored motions are primarily speed of execution and algorithmic security (by minimizing computation). The major advantages to synthesis are the reduced parameter set size (and hence less information that needs to be acquired or communicated) and the concomitant generalized motion control: walk, reach, look-at, etc. The principal disadvantages to pre-stored motion are their lack of generality

(since every joint must be controlled explicitly) and their lack of anthropometric extensibility (since changing joint-to-joint distances will change the computed locations of end effectors such as feet, making external constraints and contacts impossible to maintain) The disadvantages to synthesis are the difficulty of inventing natural-looking motions and the potential for positional disaster if the particular parameter set or code should have no solution, fail to converge on a solution, or just compute a poor result In particular, we note that inverse kinematics is not in itself an adequate model of human *motion* – it is just a local positioning aid [6, 34] The issue of building adequate human motion synthesis models is a wide open and complex research topic

Since accurate human motion is difficult to synthesize, motion capture is a popular alternative, but one must recognize its limited adaptability and subject specificity Although a complex motion may be used as performed, say in a CD-ROM game or as the source material for a (non human) character animation, the motions may be best utilized if segmented into motion “phrases” that can be named, stored, and executed separately, and possibly connected with each other via transitional (non captured) motions [17, 51] Several projects have used this technique to interleave “correct” human movements into simulations that control the order of the choices While 2D game characters have been animated this way for years – using pre-recorded or hand animated sequences for the source material – recently the methods have graduated to 3D whole body controls suitable for 3D game characters, real time avatars, and military simulations that include individual synthetic soldiers [47, 29, 13]

2.2 Control for Interactivity

Whichever motion generation technique is used, there must be a way of triggering the desired activity in the avatar Specifying the motion can be as simple as direct sensor tracking (where each joint is driven by a corresponding sensor input), end effector tracking (where inverse kinematics or other behaviors generate the “missing” joint data), or external invocation via menu, speech, or button selection of the actions (whether then synthesized or interpreted from pre-stored data) The interesting observation is that the only mechanism available to an “unencumbered” participant is actually speech! Any other avatar control mechanism requires either a hands-on device (mouse, keyboard, glove input), or else external sensors and a limited field of movement While there is considerable progress in using computer vision techniques to capture human motion [3, 26, 23, 31], both user mobility and movement generality are still in the future

Our intention is not to promote speech input *per se*, but to use this observation to promote (in Section 3 a *language-centered* view of action “triggering” augmented and elaborated by parameters modifying lower-level motion synthesis or playback (For example, this technique is used to great advantage in virtual environment applications such as the immersive interface to **MediSim** [56] and in the responsive characters in **Improv** [45, 46]) Although textual instructions can describe and trigger actions, details need not be explicitly communicated

Thus the agent/avatar architecture must include semantic interpretation of instructions and even a lower reactive level within the movement generators that allows motion generality and environmental context-sensitivity

2.3 Control for Autonomy

Providing a virtual human with human-like reactions and decision-making is more complicated than controlling its joint motions from captured or synthesized data. Here is where we engage the viewer with the character's personality and demonstrate its skill and intelligence in negotiating its environment, situation, and other agents. This level of performance requires significant investment in decision-making tools. We presently use a two level architecture

- to optimize reactivity to the environment at the lower level (for example, in the choice of footsteps for locomotion through the space) [49, 33, 14],
- to execute parametrized scripts or plan complex task sequences at the higher level (for example, choosing which room to search in order to locate an object or another agent, or outlining the primary steps that must be followed to perform a particular task) [41, 7]

The architecture is built on Parallel Transition Networks PaT-Nets [6]. Nodes represent executable processes, edges contain conditions which when true cause transitions to another node (process), and a combination of message passing and global memory provide coordination and synchronization across multiple parallel processes. Elsewhere we have shown how this architecture can be applied to the game of "Hide and Seek" [7], two person animated conversation ("Gesture Jack") [18], simulated emergency medical care (MediSim) [20], a real-time animated "Jack Presenter" [42], and multi-user "JackMOO" virtual worlds. Currently we are using this architecture to construct appropriate gestural responses from a synthetic agent, create appropriate visual attention during high-level task execution, manage locomotion tasks, and study multi-agent activity scheduling.

2.4 Gesture Control

Human arms serve (at least) two separate functions: they permit an agent/avatar to change the local environment through dextrous activities by reaching for and grasping (getting control over) objects [28, 25], and they serve social interaction functions by augmenting the speech channel with communicative emblems, gestures and beats [18].

For the first function, a consequence of human dexterity and experience is that we are rarely told how to approach and grasp an object. Rather than have our virtual humans learn – through direct experience and errors – how to grasp an object, we provide assistance through an object-specific relational table (OSR). Developed from ideas about *object-specific*

reasoning [37], the OSR has fields for each graspable site (in the *Jack* sense of an oriented coordinate triple) describing the appropriate handshape, grasp approach direction, and most importantly, its function or purpose. The OSR is manually created for graspable objects and allows an agent to look up an appropriate grasp site given a purpose, use the approach vector as guidance for the inverse kinematics directives that move the arm, and know which handshape is likely to result in reasonable finger placement. The hand itself is closed on the object through local geometry information and collision detection.

The second function of gestures is non-verbal communication. Thus gestures can be metaphors for actual objects, give indicators (via pointing) of location or participants in a virtual space around the speaker, or augment the speech signal with beats for added emphasis [18]. Currently we are working on embedding culture-specific and even individual personality gesture variations. The potential interference between practical and gestural functions is leading to a resource-based priority model to resolve conflicts.

Given that arm control for avatars requires fast position and orientation of the hands for either reaching or gestural function, fast computation of arm joint angles is essential. In recent work we have pushed beyond iterative inverse kinematics [61] to analytic formulas that can easily keep up with a live performance or a motion synthesizer outputting end effector position and orientation streams [58]. By extending this idea to the whole body, multiple individuals (3-10 on an SGI RE2) may be controlled in real-time by arbitrary end-effector and global body data alone [62].

2.5 Attention Control

A particularly promising connection is underway to connect **PaT-Nets** into other high level "AI-like" planning tools for improved cognitive performance of virtual humans. We have shown how an autonomous agent can be controlled by a high level task modeler ([24]), and how some important human motor behaviors can be generated automatically from the action requests. As tasks are generated for the *Jack* figure, they are entered into a task queue. An *attention resource manager* [21] scans this queue for current and future visual sensing requirements, and directs *Jack's* eye gaze (and hence head movement) accordingly. For example, if the agent is being told to "remove the power supply," parallel instructions are generated to locomote to the power supply area and attend to specific visual attention tasks such as searching for the power supply, scanning for potential moving objects, and periodically watching for obstacles near the feet. Note that normally none of this attentional information appears explicitly in the task-level instruction stream, yet attentional and sensing actions consume finite amounts of time and accordingly pace other actions.

2.6 Locomotion with anticipation

In order to interact with a target object, an agent must determine that it is not within a suitable distance and must therefore locomote to a task-dependent position and orientation prior to the initiation of the reach and grasp. Such a decision is readily made by embedding it in a PaT-Net representing potential actions that enable the specified action. Moreover, the locomotion process itself uses the two level architecture: at the lowest level the agent or avatar gets a goal and an explicit list of objects to be avoided, the other level encapsulates locomotion states and decisions about transitions. For example, the agent could be walking, hiding, searching, or chasing. If walking, then transitions can be based on evaluating the best position of the foot relative to the goal and avoidances. If hiding, then assessments about line of sight between virtual humans are computed. If searching, then a pattern for exhaustively checking the local geometry is invoked. Finally, if chasing, then the goal is the target object, but if the target goes out of sight, the last observed position is used as an interim goal. These sensing actions and resulting decisions are captured in the LocoNet [48].

2.7 Multi-agent task allocation

By encapsulating virtual human activities in PaT-Nets, we can interactively control the assignment of tasks to agents. A menu or program binds actions to individuals, who then execute the PaT-Net processes. Since the processes have the power to query the environment and other agents before starting to execute, multi-agent synchronization and coordination can be modeled. Thus an agent can start a task when another signals that the situation is ready, or one agent can lead another in a shared task. The latter would be especially useful when an avatar works with a simulated agent to perform a two-person task. One virtual human is designated as the "leader" (typically the avatar, so the live participant is in control) and the other the "follower". The follower's timing and motion are performed after each time-stepped motion of the leader. (The reverse situation, where the agent leads the avatar, may be needed for training and educational applications.) These are clearly the first steps toward a virtual social architecture.

We developed a prototype system for agent task assignment to evaluate a multi-function aircraft maintenance equipment cart ("MASS"). The user specifies tasks for an agent, and the agent accepts tasks for which it is both qualified and responsible. The tasks can be queued in advance, and are executed as prior tasks are completed or other agent or environment conditions obtain.

Once we can generate and control multiple agents and avatars, many social and community issues arise including authentication of identity, capabilities, permissions, social customs, transference of object control, sharing behaviors, coordinating group tasks, etc. Underlying technology to share interactive experience will depend on distributed system protocols and communication technology, client workstation performance, avatar graphics, and so on. Many of these issues are being addressed by other *ad hoc* groups, such as *Living Worlds* [38],

Open Community [43], and *Universal Avatars* [59] Having two avatars “shake hands” is considered the first stage of a social encounter requiring significant attention to the details of avatar interaction, body representation, and action synchronization Assuming that the communications can be done fast enough (a big assumption), our avatars should be able to reach for each other’s hand, detect a collision/connection, and then allow the follower avatar to position his/her hand according to the leader’s spatial position Indeed, such a demonstration has already been readily constructed by Stansfield at Sandia National Labs with *Jack* avatars, in-house network communication software, head-mounted displays, and end effector position/orientation sensors on the participants Handshaking between virtual agents is discussed in the context of *Improv* [46] Agent and avatar handshaking is considered in *JackMOO* (Section 4)

3 Connecting Language and Animation

Even with a powerful set of motion generators, a challenge remains to provide effective and easily learned user interfaces to control, manipulate and animate virtual humans Interactive point and click systems such as *Jack* work now, but with a cost in user learning and menu traversal Such interfaces decouple the human participant’s instructions and actions from the avatar through a narrow and *ad hoc* communication channel of hand and finger motions A direct programming interface, while powerful, must be rejected as an off-line method that moreover requires specialized computer programming understanding and expertise The option that remains is a language-based interface

Perhaps not surprisingly, instructions for people are given in natural language augmented with graphical diagrams and occasionally, animations Recipes, instruction manuals, and interpersonal conversations use language as the medium for conveying process and action While our historic interest in instructions has been on creating animations from instructions [8, 6, 60], we have recently begun to examine the inverse process, namely, generating text from the **PaT-Net** representations of animations The purpose is primarily to help automate the production of aircraft maintenance instruction orders (manuals) in conjunction with the animation of the tasks themselves The expectation is that the synthesized *text* material ought to reflect the proper execution of the tasks (which can be *visually* verified through the animation) and will have consistency across the entire document By the same principles, being able to process the textual instructions will aid in discovering ambiguities, omitted steps, or inappropriate terminology

The key to linking language and animation lies in constructing *Smart Avatars* that *understand what we tell them to do* This requires a semantic representation of actions, objects, and agents which is simultaneously suitable for execution (animation) as well as natural language expression We have called this *implementable semantics* the representation must have the power of a (parallel) programming language which drives a simulation (in a context of a given set of objects and agents), and yet supports the enormous range of expression,

nuance, and manner offered by language. We consider two aspects of this problem in the remainder of this paper. The first part (Section 4) considers a 3D avatar extension called *JackMOO* to an existing lambdaMOO multi-user system. The second (Section 5) constructs a draft specification for a Parameterized Action Representation (PAR) which uses PaT-Nets as an implementation language [9].

4 JackMOO

³ Creation of compelling 3-dimensional, multi-user virtual worlds for education and training applications requires a high degree of realism in the appearance, interaction, and behavior of avatars within the scene. Development and deployment of systems supporting multi-user, 3-dimensional virtual worlds are today enjoying some commercial success. Systems as Alpha World [1], Pueblo [19], and Black Sun [16] provide facilities of various levels of sophistication to select an avatar, move about a virtual world, and interact with other avatars and objects they may find.

We have produced a prototype system, *JackMOO*, which combines *Jack* and LambdaMOO [22], a multi-user, network-accessible, programmable, interactive system which has been used for the construction of text-based conferencing, educational/training, and other collaborative software. *JackMOO* allows us to store the richer semantic information necessitated by the scope and range of human actions that an avatar must portray, to express those actions in the form of natural, imperative sentences. *JackMOO* therefore provides us with an testbed for language control of avatar animation. This section describes *JackMOO* and its components, especially a *JackMOO* client program that mediates the flow of control between *Jack* and the lambdaMOO server and provides the primary user interface to the system.

Of central importance to *JackMOO* is the association of human action verbs with possibly several PaT-Nets that realize the action on the virtual human on the *Jack* display. Actions as *step-forward*, *turn-around*, and *look-at*, are specified in the *Jack* environment in the form of executable programs, providing the level of interface necessary for the control of a virtual human avatar in a virtual world. PaT-Nets thus function as a high-level API accessing underlying *Jack* behavior and functionality.

4.1 LambdaMOO

LambdaMOO is a network-accessible, multi-user, programmable, interactive system well suited for the construction of text-based conferencing systems, educational/training systems, and other collaborative software. LambdaMOO's roots are in multi-user, collaborative game

³Based on a paper by T. J. Smith, J. Shi, J. P. Granieri, N. I. Badler, *JackMOO: A web-based system for virtual human simulation*. Presented at WebSim'98, San Diego, CA, January 1998.

environments in which users are represented by a character possessing certain distinguishing attributes are placed within a virtual reality where they may interact with other users and objects that may be encountered

The interface to a lambdaMOO world is text-based. Users communicate with the world by typing one line commands resembling imperative English sentences, e.g., *take the ball, strike dragon on the head, etc.* Such commands are parsed and executed by the system and result in changes in the virtual environment, such as location of the user's character, the appearance of some object, interactions with other users, etc. Such changes are communicated to the user by means of text. *You pick up the ball, The dragon appears to be angry and is coming toward you menacingly!* etc. In addition, each of the other players in the user's neighborhood receive text notification of the user's actions. *John has picked up the ball, The dragon is angrily advancing on Norm, etc.* Users may interact with each other on at least two levels: they may *talk* to one or all of the other players and they may *emote*, i.e., reveal emotions to one or more players. *John laughs at Norm's predicament, Sandy gives Norm a hug, etc.*

LambdaMOO consists of two major components: the *server* and the *database*. The server is a program written in a standard programming language that manages multiple network connections, maintains queues of commands to be executed, controls all access to the database, and executes programs written in the LambdaMOO programming language.

The database contains objects representing all components of the virtual reality. Each object consists of properties containing data intrinsic to the object and verbs, programs written in the LambdaMOO programming language that provide the objects with their particular behaviors. A LambdaMOO virtual world is organized as a series of room objects, connected by *entrance* and *exit* objects. Rooms provide the containers for other objects in the world: players, TV sets, documents which may be read, containers holding additional objects, etc. Players, themselves objects, move about the lambdaMOO world, room by room, interacting with objects and other players they may encounter.

Objects are organized into single-inheritance hierarchies. When objects are created they *inherit* the properties and verbs of their ancestors. Additional properties and verbs as well as *specializations* of inherited components may be defined to give the new object its unique behavior and appearance. In particular, the *JackMOO* world contains a specialization of the LambdaMOO generic room object called a *Jack Room*. This object contains properties and verbs that implement the interface between the LambdaMOO server and the remainder of the *JackMOO* system. Each LambdaMOO room that has a *JackMOO* realization is a specialization (a child of) of the *Jack Room* object containing, in particular, properties that contain scene information to be loaded into *Jack* for the room and its contents to be visualized. Specializations (children of) of the LambdaMOO generic player object, the *Jack Player*, provide the properties and verbs necessary to define and control the individual avatar representation for a player in the *JackMOO* system. Default avatar properties are provided in the parent *Jack Player* object for those *JackMOO* users having no unique avatar representation.

The interface to the *JackMOO* system is activated when a *Jack Player* enters a *Jack Room*. Verbs and properties defined on the instance of the *Jack Player* cause the new arrival's avatar to appear on the *Jack* displays of all of the room occupants who are *JackMOO* users, and to display each *JackMOO* occupant's avatar on the new arrival's *Jack* screen. Avatar positional and orientation information saved in the database is used to locate the avatars "where they belong" in the scene. Movement and control of the avatar is done by executing appropriate verbs on the *JackMOO* objects through the *JackMOO* client.

Each of the *JackMOO* avatar behavior verbs have the following responsibilities:

- Update the persistent information maintained for the avatar, e.g., change its stored position when a "move" command is received.
- Perform any actions with other objects in the room that may be affected by the verb. If my avatar picks up the red ball, the ball moves from its original location to my avatar's hand.
- Issue commands to the *Jack* component of the system to display avatar changes to the end user.
- Broadcast changes to all of the connected *JackMOO* clients. This provides the synchronization of each of the *JackMOO* displays as the avatar moves around and interacts in the *JackMOO* world.
- Broadcast a textual description of the action and its effect on the environment to all players in the room. *JackMOO* can thus be accessed as a *standard*, text-only lambdaMOO world.

4.2 JackMOO Client

The user interface to *JackMOO* is provided by a client program written in Java running inside Netscape. The Client's responsibilities include:

- Establishment and maintenance of connections to LambdaMOO and *Jack*.
- Sending user input to LambdaMOO.
- Routing text emanating from LambdaMOO to either *Jack* or the Client's text output area.

The Client provides text-based LambdaMOO commands typed by the user into a text input area. The Client provides a MOO output area, a scrolled window in which textual output from LambdaMOO is displayed. This is managed by a connection established by the Client to the LambdaMOO server.

Commands to *Jack* are sent from LambdaMOO to the Client in a special text string called the *JackMOO* protocol. The protocol identifies a text string as one that should not be displayed on the Client's MOO output area - but should be sent to *Jack* to effect the motion of the avatar on the *Jack* display screen. Text coming from LambdaMOO without the protocol is displayed in the Client's MOO output area. Thus, the Client functions as a router for messages emanating from the LambdaMOO server - sending commands to *Jack* or displaying as text in its MOO output area.

The demonstration includes three experiments

- 1 Handshaking scenario, showing avatar-avatar interaction and synchronization through PaT-Nets
Agent-Object-Preposition interaction showing walking *around* a chair, arbitrarily placed
- 3 Agent-agent relationship showing one agent accepting a *follow me* command from another

These and other examples show that smart avatars can portray interesting, context-sensitive actions based on real-time execution of textual commands

5 Parameterized Action Representation

⁴Processes are ⁵

- 1 A system of operations in the production of something
- 2 A series of actions, changes, or functions that bring about an end or result
- 3 Course or passage of time
- 4 Ongoing movement, progression

So processes have two general forms - one in which something is *happening* and another in which something is *completed*. The conventional terminology is that the latter are *culminated processes*. We will also term any non-culminated process *active* to clearly distinguish this case. It is therefore convenient to graphically present processes as *nodes* in which some "action, change, or function" takes place, and *arcs* which link one process (node) to another

⁴This section is taken from N. Badler, B. Webber, M. Palmer, T. Noma, M. Stone, J. Rosenzweig, S. Chopra, K. Stanley, J. Bourne, and B. Di Eugenio. Final report to Air Force HRGA regarding feasibility of natural language text generation from task networks for use in automatic generation of Technical Orders from DEPTH simulations. Technical report, CIS, University of Pennsylvania, 1997.

⁵American Heritage Electronic Dictionary, Copyright 1991 by Houghton Mifflin Company

```

type parameterized action =
  (agent agent representation,
   objects sequence object representation,
   applicability conditions sequence disjunctive-queries,
   culmination conditions sequence disjunctive-sensor-queries,
   spatiotemporal spatiotemporal specification,
   manner manner specification,
   subactions actions)

```

Figure 1 The parameterized action type

that temporally follows either by virtue of culmination of the first or other circumstances. A process can be recursively defined as a network (or graph) of process nodes (possibly disconnected, i.e. parallel). Thus, a hierarchy of processes can exist, grounding out at single process nodes for the simplest types of processes.

An *action*⁶ is just a particular kind of process which involves a volitional agent acting in the world. We call our representations of actions *Parameterized Action Representations* (PARs) and they contain a necessary slot for an agent. A generic process representation is a PAR with an optional agent slot. In this report, we deal almost exclusively with PARs that contain the necessary agent slot.

Our representation is a modified version of the representation used by Kalita and Lee [32], expanded to include culmination conditions, agent/object representations, as well as more detail about the specifics of actions.

5.1 Parameterized Action Representation Specification

The top-level type in the representation is the *parameterized action* (see Figure 1), we call it “parameterized” because an action depends on its participants (agent and objects) for the details of how it to be accomplished. For instance, opening a door and opening a window will involve very different behaviors on the part of the agent. The subparts of a parameterized action can refer to particular aspects of the agent and objects as part of their meaning. In the sections that follow, we specify and discuss each subpart of the representation.

5.1.1 Agent

As mentioned above, the agent is the distinguishing feature between an action and a mere process. It specifies which agent is carrying out the process described in the rest of the

⁶Defined by the American Heritage Electronic Dictionary as “The state or process of acting or doing”

representation. We assume that the agent refers to a human model or a physical force like gravity (in which case the agent is understood to be causal and not volitional).

The agent type (see Figure 2) and the object type (see Figure 3 in Section 5.1.2) represent agents and objects respectively. They are very similar in concept except that the agent type has some extra fields which also describe the behaviors of the agent which would influence some of the actions of the agent.

For each instance of the agent type, a list of actions that the agent is capable of performing is specified. The agents can also be considered to be capable of playing different roles. For each role, the agent performs different actions. So, instead of maintaining one long list of actions, we could group these actions under different roles. For example, the actions involved while driving a car like grasp a steering wheel, sit with foot on the accelerator pedal, etc., would be grouped under the "car-driver" role. Each of the listed actions is a primitive action. Unlike for the objects, each action is associated with a set of applicability conditions (test for reachability, etc) which check if the action can be performed by the agent. If not, another set of primitive actions is generated for that agent which have to be completed before the current action can be performed. The agent type also has a field for specifying nominal values and the distribution type and range for some of the actions and state space descriptors. For example, the walking rate of the agent could be specified to have a nominal value of 2 steps per second with a normal distribution and standard deviation of 1. This gives a range of values over which the walking rate can be varied.

5.1.2 Objects

The object type is defined explicitly for a complete representation of a physical object (see Figure 3). Each object in the environment is an instance of this type. We could also distinguish between objects and causal models.

The *state* field of an object describes a set of constraints on the object which leave it in a default state. The object continues in this state until a new set of constraints are imposed on the object by an action which causes a change in state. The other important fields are the reference coordinate frame, a list of grasp sites and directions defined with respect to the object.

For each instance of the object type, a set of actions are defined. Each of these actions can be further described as a group of one or more actions. Also, the objects can be represented hierarchically. This allows us to describe actions for a class of objects rather than for every object. The actions are defined at the highest possible level in the object tree. So the action field in an instance of the object type could point to a description or to the parent.

5.1.3 Applicability Conditions

The applicability conditions of an action specify what needs to be true in the world in order to carry out an action. These can refer to agent capabilities, object configurations, etc., and

```

type agent representation =
    (coordinate-system site,
     state state space,
     rel-dir relative directions,
     special-dir special directions,
     grasp-sites sequence site,
     capabilities sequence actions-and-applicability,
     nominals sequence value-ranges)

type actions-and-applicability =
    (action parameterized action,
     applicability sequence disjunctive-queries)

type value-ranges =
    (var powerset parameter,
     mean powerset var-types,
     standard deviation powerset var-types,
     distribution powerset distribution)

type parameter =
    (id string)

type var-types =
    (real, real vector, integer)

type distribution =
    (normal, poisson, uniform)

```

Figure 2 The agent representation type

```

type object representation =
  (coordinate-system site,
   state state space,
   rel-dir sequence relative direction,
   special-dir sequence special direction,
   grasp-sites sequence site,
   actions sequence parameterized action)

type site =
  (position real vector,
   orientation real vector)

type state space =
  (position real vector,
   velocity real vector,
   acceleration real vector,
   force real vector,
   torque real vector)

type relative direction =
  (name (front, back, left, along, inside),
   value real vector)

type special direction =
  (name string, value real vector)

```

Figure 3 The object representation type

```

type disjunctive-queries =
    (query, sequence query)

type query =
    (object-query, agent-query, global-query)

type object-query =
    (obj object, oquery string)

type agent-query =
    (agent agent description, aquery string)

type global-query =
    (variable global variable, gquery string)

```

Figure 4 Applicability conditions as a series of *disjunctive-queries*

are represented by conjunctions of disjunctions of queries on the state of objects, agents, and global variables (see Figure 4). For instance, the applicability conditions for changing a light-bulb could be represented as *has(agent, light bulb) AND (can_reach(agent light-fixture) OR has_ladder(agent))*

5.1.4 Culmination Conditions

Whether an action is considered a process or a culminated process depends on whether a culmination condition (default or otherwise) is specified. Processes do not have culminations associated with them, that is, processes just end with no consequences attached to their ending. Culminated processes, on the other hand, can only be said to have occurred or be completed if they have reached their culmination, in which case the consequences are that the culmination conditions hold.

Culmination conditions, or the conditions that will hold when an action is completed, are also a series of queries like applicability conditions, however they are restricted to queries of the agent's sensors (see Figure 5). That is, the only way an agent is allowed to determine information about the world is through its sensors. For instance, an agent cannot simply query an object, say the lid of a jar, to see if it is loose; the agent must query a sensor to find out if the lid is loose. So, instead of a query such as *loose(lid)*, the agent must use a query such as *sense(loose(lid))* in order to determine whether the culmination condition in the action "Turn lid until loose" holds. Such sensing processes are needed independently of the need to represent culmination conditions (see Section 5.2.2).

```

type disjunctive-sensor-queries =
    (sensor-query, sequence sensor-query)

type sensor-query =
    (agent agent description, squery string)

```

Figure 5 Culmination conditions as a series of disjunctive-sensor-queries

5 1 5 Spatiotemporal Specification

At an abstract level, basic actions involve manipulating spatial or geometric relations between objects (or possibly between the agent and objects). Thus, part of the spatiotemporal specification of an action can be a *spatial goal*, which consists of the type of the manipulation (establishing, terminating, maintaining, or modifying) and the relation that is being manipulated (see Figure 6). The relation is expressed as queries, which will refer to spatial predicates on objects and agents. For instance, the action in "Put the block on the table" would have the spatial goal of establishing the relation of the block being on the table. The relation could be expressed as a query to the block, "block on(table)", which would return a "yes" or "no" answer. The relation would be considered established when the query returned a "yes". In order to convert these abstract spatial goals into the lower level kinematics and dynamics, an algorithm is needed that can understand the queries (e.g., what it means for something to be "on" something else) and convert them into motions for establishing, terminating, etc., the relations they express.

The spatial goals are provided by the lexical terms used in the input or, conversely, are formed by recognizing certain changes in directions or relationships in the object state. In either case we consider a set of terms including those considered and defined in Badler's earlier work [4].

across	after	against	ahead-of	along
apart	around	away	away-from	back
back-and-forth	backward	behind	by	clockwise
counterclockwise	down	downward	forward	from
in	in-the-direction-of	into	inward	off
off-of	on	onto	onward	out
out-of	outward	over	sideways	through
to	together	toward	under	up
up-and down	upward	with		

The semantics of these terms for animation synthesis is to be implemented through **PaT-Nets** that transform the given parameters (e.g., objects involved) into movement paths,

```

type spatiotemporal specification =
  (goal spatial-goal,
   kinematics kinematics specification,
   dynamics dynamics specification,
   frequency rate specification)

type spatial-goal =
  (execution-type (establish, terminate, maintain, modify),
   relation sequence disjunctive-queries)

type kinematics specification =
  (position real vector
   time time specification,
   velocity real vector,
   acceleration real vector,
   path path specification)

type dynamics specification =
  (force real vector,
   torque real vector)

type rate specification =
  (period real vector
   amplitude real vector)

type path specification =
  (direction (relative direction, special direction, real vector),
   scale real,
   pathpoints sequence 3Dpoints)

```

Figure 6 The spatiotemporal specification type

vectors, or directions eventually processed by the primitive actions sent to the agent. For textual synthesis, PaT-Net “recognizers” as defined in [4] can be executed to vote for the semantic term most closely describing the changing situation being presented. Although many terms are simply recognized from directions relative to object labels (“forward” from movement with front of object in the lead, etc.), others require a multi-node PaT-Net to determine the sub-steps and completion of the activity (e.g. across, around, back and-forth, etc.)

Mathematically, a motion can be represented in any one of the three domains - kinematic, dynamic and frequency. The motion generating primitive functions take parameterized inputs to generate the exact motions. The modifiers impose additional constraints on the motion. At any time, one or more of the constraints are active. The constraints can be specified in any of the above mentioned three domains. The mathematical components of the representation are designed to facilitate conversion of movements defined or specified in one system. Conversion procedures exist to change kinematics data into dynamics information (*inverse dynamics*) and vice versa (*forward dynamics*) [35]

The basic parameters for modifying a motion in the kinematic domain are position, time, velocity, acceleration and path. These parameters are in general relative but they could also be global. In the dynamic domain, given a motion and the agent/object involved, it is possible to compute the forces and torques at the joints required to generate the given motion. If an object is able to rotate about an axis, then a force applied in a direction perpendicular to the axis of rotation causes the object to rotate. The speed of rotation is governed by the magnitude of the force. So, to modify the motion in the dynamic domain, we can specify relative forces and torques. In the frequency domain, we can represent the motion as a function of time using Fourier series expansion. The parameters to vary the motion in this domain could then be mapped to period and amplitude.

5.1.6 Manner Specification

At the lexical level is an enumeration of manner adverbs related to verbs. At the action representation level, we have the “manner” or “Effort-Shape” (ES) parameters described below. To go from the lexical level to the action representation level, we would have procedures that operate on the (verb, adverb) pairs to translate them into manner affecting ES parameters. One set of adverbs will modify the ES parameters directly, taking the default values from the stored nominal values. So an instruction to walk quickly would involve looking up the nominal walk speed and invoking a generic procedure “quicker” that scales the nominal rate by a factor of say 50% toward the maximum walking rate. This scaled rate can now be filled in the agent’s current forward motion rate and used by the locomote action. A second class of adverbs could be interpreted as members of the first with some assumptions. For example, “carefully” can be interpreted as slowly and with low accelerations (something that can be consumed by the ES notation). In some cases, then, an adverb tells what movement *pattern*

```

type manner specification =
    (effort effort parameters,
     shape shape parameters)

type effort parameters =
    (space real range[-1,1],      /* Indirect = -1, Direct = 1 */
     weight real range[-1,1],    /* Light = -1, Strong = 1 */
     time real range[-1,1],      /* Sustained = -1, Sudden = 1 */
     flow real range[-1,1])      /* Free = -1, Bound = 1 */

type shape parameters =
    (vertical real range[-1,1],   /* Sinking = -1, Rising = 1 */
     lateral real range[-1,1],    /* Narrowing = -1, Widening = 1 */
     sagittal real range[-1,1])  /* Retreating = -1, Advancing = 1 */

```

Figure 7 The manner specification type

to apply to a (novel) situation. The patterns would be (pre-stored) action (fragments) that would be newly bound to the current object and agent. So these adverbs are interpreted by procedures that take the action as a parameter, substitute new object bindings, and then compose the spatiotemporal characteristics into the current action being done.

Manner specifications describe the way in which an agent carries out an action. We define manner as composed of an Effort parameter and a Shape parameter (see Figure 7). The Effort parameters, which are derived from Effort Notation [10], express the quality of a movement. Each parameter takes on a real value in the range from -1 to 1. The Effort elements include Space, Weight, Time, and Flow. Space varies from direct (1) to indirect (-1) and describes whether a person is focusing in on his/her surrounding space or not. Weight expresses the forcefulness of a movement at its end, ranging from light (-1) to strong (1). The Time element describes the sense of urgency in the beginning of movement along the continuum between sustained (low acceleration) (-1) and sudden (high acceleration) (1). Flow describes the progression of a movement and ranges between free (dynamically-driven) (-1) and bound (kinematically-driven) (1). The Shape specification describes the general shape and pose of the agent's body, giving information on the directional goals of the agent. The Shape parameters are specified for each plane - vertical, lateral, and sagittal - as real values in the range [-1,1]. The vertical parameter varies from sinking (-1) to rising (1), lateral varies from narrowing (-1) to widening (1), and sagittal from retreating (-1) to advancing (1).

```

type actions =
    (actions sequence labeled actions,
     constraints sequence constraint-and-labels)

type labeled actions =
    (label string, action parameterized action)

type constraint-and-labels =
    (constraint (seq, par, par-join, par-indy, while),
     action-label1 string, action-label2 string)

```

Figure 8 The subactions type

5.1.7 Subactions

Complex actions, such as fueling a vehicle, will involve several subactions. Thus, the *subactions* part of the representation consists of a list of parameterized actions and temporal constraints among the actions (see Figure 8). The possible temporal relationships between two actions are

Sequential (seq) The actions are done sequentially (default)

Parallel (par) The actions are done in parallel

Jointly parallel (par-join) The actions are done in parallel and no other actions are done until after both have finished

Independently parallel (par-indy) The actions are done in parallel but once one of the actions is finished, the other one is stopped

While parallel (while) The subordinate action is done while the dominant action is done, once the dominant action finishes the subordinate action is stopped

A complex action can be considered done if all of its subactions are done or if its explicit culmination conditions are satisfied

5.2 Action Representation to PaT-Nets Control Algorithm

In order to produce animation, actions represented in the manner described above must be converted into PaT-Nets. All the actions of an agent which correspond to a given set of instructions are referred to as the top-level actions and are maintained at the highest level in a queue tree. Each of these high level actions might have subactions. All these

subactions are now maintained in a queue at the next level. For every action, a **PaT-Net** is spawned. For every high level action, the subactions form the children and the higher level action is assumed completed only after all the children's actions are completed. An action is also considered completed if the culmination conditions of some higher level **PaT-Net** are satisfied. A sequence of actions is maintained as children from left to right, the leftmost child being executed first. Once an action is completed, the action on its right is then considered. However, if any set of actions have to be executed in parallel, we use the concepts of *par-join*, *par-indy*, etc (see Section 5.1.7)

Given an action specification from the top-level queue, there are three possibilities

- 1 The action is a primitive action and is added to a lower level queue. Once there, the applicability conditions for the action are checked. If the conditions are true, then a **PaT-Net** that corresponds to the primitive action is spawned for the execution of the action. If the conditions do not hold, another list of subactions may be added as children of the current action node in the queue tree. (See Section 5.2.1 for more on primitive actions.)
- 2 Object-specific information, such as an action definition defined specifically for an object involved in the action or directions relative to an object, etc, needs to be obtained. The expansion of the action with object-specific information is added to the queue tree below the current action node, including any subactions. (Whenever the object or agent is accessed for information on the action, it is first checked if it has been defined for that agent/object, possibly higher in the object hierarchy. If not, then an error message is generated.)
- 3 The action can be expanded into subactions. Each of these subactions then forms a child of the current action node in the queue hierarchy.

See Figure 9 for a simplified version of the control algorithm which ignores most of the details of the queue hierarchy that is maintained.

5.2.1 Primitive Actions

Currently, we refer to the basic actions in *Jack* as the primitive actions. These include actions such as move (which includes translate and rotate within it), grasp, reach, locomotion, visual search, task appropriate attention, etc. Each of these primitive actions can be generated by various methods like motion scripting, live motion capture, various algorithms using the principles of dynamics and control, fast inverse-kinematics, random noise, walking algorithms, etc. Instead of generating motions for the whole agent (we consider only the human agent for this purpose) using only one of these sources at a time, we could use a different source for each part of the body. For this purpose, currently, at a higher level, the body of the human model is divided into parts: head, arms, torso, legs, and figure position.

```

For each parameterized action, Act, in the Instruction-queue
  Initialize Q to be an empty queue
  If Act subactions is non-empty then
    Put each element in Act subactions into Q
  Else put Act into Q
  While Q is not empty
    Let A be the element at the head of the queue
    If A is a primitive action then
      Let P be the \patnet\ corresponding to A
      Put P into the Primitive-queue
    Else if A needs object-specific expansion then
      Let L be the result of object-specific expansion on A
      Put each element of L into Q
    Else if A subactions is non-empty then
      Put each element of A subactions into Q

```

Figure 9 Action Representation to PaT-Nets Control Algorithm

A different PaT-Net is used for each motion generator. The parameters to the motion generator would then be the part of the body that they would control. If more than one motion generator were to control the same part of the body, it could be resolved in two ways - blend (or any other binary operation) the resulting motions together or set up a priority for the motion generators and arbitrate appropriately between them.

5.2.2 Sensing processes

An important contribution of the *Jack* virtual human model is that it allows us to view and analyze human actions such as locomotion, reach, grasp and gesture. Tasks composed of such elementary actions may in turn be combined and performance observed. Simply stringing together sequences of tasks, however, is unrealistic since humans require time to visually attend and acquire information before or during execution of tasks. The action representation requires a framework of sensing processes in which visual search, hand-eye coordination and attention guides the sequencing and simulation of actions, especially motor tasks. Other sensing activities that we must represent are monitoring, probing the state of the world via global or internal variables, checking time on global or relative clocks and evaluating spatial/geometric predicates (relationships).

We contend that sensing activities or processes are themselves tasks whose duration and execution are significant. Further, timing information for processes such as attention and visual search is not static. Such information will vary with cognitive load, expertise

and *type of elementary action* Studies show that eye movement latencies are longer when accompanied by certain spatially oriented gestures like pointing[15]

The *Jack* system maintains a geometry of objects and relevant sites in a virtual world Since access to these objects is *negligible* in terms of computer simulation time, some explicit representation of sensing or investigative process is particularly important Further efforts are required to model the time durations introduced by sensing and attentive processes Since it is likely that a comprehensive attentional timing model would be difficult to develop, we may first incorporate simplified reaction time computations to achieve a modest level of performance veracity

The sensing processes themselves include a number of particular features

- Spawn, if necessary, a sub-net with attentional and/or manipulative and/or cognitive actions
- Check time on global or relative clocks
- Send messages to probe external global variables
- Send messages to probe specific internal states of other processes
- Evaluate spatial/geometric predicates (pseudo-pattern recognition for relationships, situational awareness, etc)
- Include sampling frequency (independent of simulation clock ticks)

To the extent made possible by the functional definitions, the *future* variable values of a process node may be queried by a sensing process In the case of a formula for linear motion, for example, the present trajectory may be extrapolated to yield an estimate of possible future location The internal object description includes fields for velocity and acceleration as well as position and orientation It is speculated that the ability to successfully run processes in a predictive mode will improve agent/agent and agent/object interactions by changing "pursuit" situations into "anticipation" situations [48]

5 2 3 Queue of Primitive Actions and Visual Attention

Requests for primitive actions are placed on a queue A queue manager is maintained that consumes such requests and determines which agent resource should be used to execute the request For example, the queue manager may determine which hand should be used to execute a grasp based on following criteria. which hand is empty, which hand is closer to the object to be grasped and also potentially which hand the agent favors for object manipulation (e g , is the agent right or left handed?)

A queue of action requests facilitates the animation of sensory procedures Such procedures may be executed parallel or in addition to motor actions For example, task related

attention is supported in our human model. The queue manager automatically invokes the appropriate attentional behavior for each *type* of task on the queue. While the agent is walking to a goal, for example, he will look at the goal and occasionally glance at his feet (when a memory uncertainty threshold is exceeded or when the agent is in close proximity to an obstacle). When grasping an object, the queue manager invokes an attentional process that determines which sites are relevant for the grasp and directs attention appropriately. Our aim is to direct attention as autonomously as possible and based on analysis of task mix. Since a queue allows lookahead of downstream tasks, we allow the potential to *interleave* or *anticipate* where attention may be directed.

6 Conclusions

This exposition has described virtual human modeling and control, with an emphasis on real-time motion and language-based interfaces. In particular, we discussed such issues as appearance and motion, interactive control, autonomous action, gesture, attention, locomotion, and multiple individuals. The underlying *Jack* architecture consists of a sense-control-act structure that permits reactive behaviors to be locally adaptive to the environment, and a PaT-Net parallel finite-state machine controller that can be used to drive virtual humans through complex tasks.

The *JackMOO* is a virtual world environment combining *Jack* with an existing multi-user technology *lambdamoo*. The *JackMOO* hybrid focuses on 3D human-like avatars and employs a language-based interface (imperative sentences) to control them. *JackMOO* provides a flexible environment in which pilot/drone and leader/follower roles may be specified and used to advantage in training and educational 3-dimensional scenarios.

We next described a first version of a Parameterized Action Representation. The PAR is meant to be the intermediate structure between simple natural language imperative sentences with complex semantics and task execution by a virtual human agent. There are many dimensions to the PAR, including slots for the agent, participating objects, applicability conditions, culmination conditions, spatiotemporal descriptions, agent manner, and suggested subactions. An algorithm for interpreting PARs within an object-oriented system was proposed, with implementation using the *JackMOO* framework in progress.

We have established a role for linguistics in knowledge modeling. Linguistic classifications have helped us by identifying typical properties and modifiers of animate agents, such as the dimensions along which agent behavior can vary. In addition, linguistic analysis can help identify typical actions of animate agents and typical modifiers for their actions. The agent's manner may be modified via Effort basis functions grounded in human motion observation. Basing an agent and action ontology on linguistic evidence and movement models ensures extensibility.

Work in progress includes

- PAR development and implementation
- Various human model improvements (walking, motion models, Effort parameters)
- Smart avatars
- Multi-user virtual environments
- Agent models with culture, roles, manner, and personality

Smart avatars and agents will need reactive, proactive, and decision-making behaviors for action execution. They will need to have individualized perceptions of context. They must understand language so that we may communicate with them as if they were real. And hardware improvements alone will not yield intelligence.

The future holds great promise for the virtual humans who will populate our virtual worlds. They will provide economic benefits by helping designers early in the product design phases to produce more human-centered vehicles, equipment, assembly lines, manufacturing plants, and interactive systems. Virtual humans will enhance the presentation of information through training aids, virtual experiences, and even teaching and mentoring. And Virtual humans will help save lives by providing surrogates for medical training, surgical planning, and remote telemedicine. They will be our avatars on the Internet and will portray ourselves to others, perhaps as we are or perhaps as we wish to be. They may help turn cyberspace into a real, or rather virtual, community.

Acknowledgments

The many students, staff, and colleagues in the Center for Human Modeling and Simulation make this effort possible. Additional information and contributors may be found through <http://www.cis.upenn.edu/~hms>

This research is partially supported by U.S. Air Force through Delivery Orders #8 and #17 on F41624-97-D-5002, Office of Naval Research (through Univ. of Houston) K-5-55043/3916-1552793, DURIP N0001497-1-0396, and AASERTs N00014-97-1-0603 and N0014-97-1-0605, Army Research Lab HRED DAAL01-97-M-0198, DARPA SB-MDA-97-2951001 (through the Franklin Institute), NSF IRI95-04372, NASA NRA NAG 5-3990, National Institute of Standards and Technology 60 NANB6D0149 and 60 NANB7D0058, SERI Korea, and JustSystem Japan.

References

- [1] Activeworlds, 1997 [http //www activeworlds com/](http://www.activeworlds.com/)
- [2] J Allbeck and N Badler *Avatars a la snow crash* In *Computer Animation '98* IEEE Press, 1998
- [3] F Azuola, N Badler, P-H Ho, I Kakadiaris, D Metaxas, and B Ting Building anthropometry-based virtual human models In *Proc IMAGE VII Conf*, 1994
- [4] N Badler *Temporal Scene Analysis Conceptual descriptions of object movements* PhD thesis, CS, University of Toronto, 1975
- [5] N Badler, M Hollick, and J Granieri Real-time control of a virtual human using minimal sensors *Presence*, 2(1) 82-86, 1993
- [6] N Badler, C Phillips, and B Webber *Simulating Humans Computer Graphics Animation and Control* Oxford University Press, New York, NY, 1993
- [7] N Badler, B Webber, W Becket, C Geib, M Moore, C Pelachaud, B Reich, and M Stone Planning for animation In N Magnenat Thalmann and D Thalmann, editors, *Computer Animation* Prentice-Hall, 1996 To appear
- [8] N Badler, B Webber, J Kalita, and J Esakov Animation from instructions In N Badler, B Barsky, and D Zeltzer, editors, *Making Them Move Mechanics Control, and Animation of Articulated Figures*, pages 51-93 Morgan Kaufmann, 1990
- [9] N Badler, B Webber, M Palmer, T Noma, M Stone, J Rosenzweig, S Chopra, K Stanley, J Bourne, and B Di Eugenio Final report to Air Force HRGA regarding feasibility of natural language text generation from task networks for use in automatic generation of Technical Orders from DEPTH simulations Technical report, CIS, University of Pennsylvania, 1997
- [10] I Bartenieff and D Lewis *Body Movement Coping with the Environment* Gordon and Breach Science Publishers, New York, NY, 1980
- [11] J Bates The role of emotion in believable agents *Comm of the ACM*, 37(7) 122-125, 1994
- [12] J Bates, A Loyall, and W Reilly Integrating reactivity, goals, and emotion in a broad agent In *Proc of the 14th Annual Conf of the Cognitive Science Society*, pages 696-701, Hillsdale, NJ, 1992 Lawrence Erlbaum
- [13] BDI-Guy, 1996 Boston Dynamics Inc , Cambridge, MA

- [14] W Becket *Reinforcement Learning of Reactive Navigation for Computer Animation of Simulated Agents* PhD thesis, CIS, University of Pennsylvania, 1997
- [15] H Bekkering, H Kigma, H Adams, A Van der Aarssen, and H Whiting Interference between saccadic eye and goal directed hand movements *Experimental Brain Research*, 106 475-484, 1995
- [16] Black Sun, Inc , 1997 [http //www blacksun com](http://www.blacksun.com)
- [17] A Bruderlin and L Williams Motion signal processing In *Computer Graphics, Annual Conf Series*, pages 97-104 ACM, 1995
- [18] 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 *Computer Graphics, Annual Conf Series*, pages 413-420 ACM, 1994
- [19] Chaco, Inc , 1997 [http //www chaco com/pueblo/index.html](http://www.chaco.com/pueblo/index.html)
- [20] D Chi, B Webber, J Clarke, and N Badler Casualty modeling for real-time medical training *Presence*, 5(4) 359-366, 1995
- [21] S Chopra Strategies for simulating direction of gaze and attention Technical report, Center for Human Modeling and Simulation, University of Pennsylvania, 1995
- [22] P Curtis LambdaMOO, 1997 Xerox PARC Ftp site [parcftp xerox com/pub/MOO](http://parcftp.xerox.com/pub/MOO)
- [23] D DeCarlo and D Metaxas The integration of optical flow and deformable models with applications to human face shape and motion estimation In *Proc CVPR*, pages 231-238 IEEE Press, 1996
- [24] S Deutsch, J MacMillan, N Cramer, and S Chopra Operator Model Architecture (OMAR) support Technical Report 8179, BBN, Cambridge, MA, 1997
- [25] B Douville, L Levison, and N N Badler Task level object grasping for simulated agents *Presence*, 5(4) 416-430, 1996
- [26] I Essa and A Pentland Facial expression recognition using a dynamic model and motion energy In *Proc of the International Conf on Computer Vision*, Cambridge, MA, 1995
- [27] M Girard and A Maciejewski Computational modeling for the computer animation of legged figures *ACM Computer Graphics*, 19(3) 263-270, 1985

- [28] J -P Gourret, N Magnenat-Thalmann, and D Thalmann Simulation of object and human skin deformations in a grasping task *ACM Computer Graphics*, 23(3) 21-30, 1989
- [29] J Granieri, J Crabtree, and N Badler Off-line production and real time playback of human figure motion for 3D virtual environments In *VRAIS Conf* IEEE Press, 1995
- [30] J Hodgins, W Wooten, D Brogan, and J O'Brien Animating human athletics In *ACM Computer Graphics, Annual Conf Series*, pages 71-78, 1995
- [31] I Kakadiaris and D Metaxas Model-based estimation of 3D human motion with occlusion based on active multi-viewpoint selection In *Proc of the Conf on Computer Vision and Pattern Recognition*, pages 81-87 IEEE Computer Society, June 1996
- [32] J Kalita and J Lee An informatl semantic analysis of motion verbs based on physical primitives *Computational Intelligence*, 1996
- [33] H Ko and N Badler Animating human locomotion in real-time using inverse dynamics, balance and comfort control *IEEE Computer Graphics and Applications*, 16(2) 50-59, March 1996
- [34] Y Koga, K Kondo, J Kuffner, and J -C Latombe Planning motions with intentions In *ACM Computer Graphics, Annual Conf Series*, pages 395-408, 1994
- [35] E Kokkevis, D Metaxas, and N Badler User-controlled physics-based animation for articulated figures In *Computer Animation Conf Proc* , 1996
- [36] D Kurlander, T Skelly, , and D Salesin Comic chat In *ACM Computer Graphics, Annual Conf Series*, pages 225-236, 1996
- [37] L Levison *Connecting planning and acting via object-specific reasoning* PhD thesis, CIS, University of Pennsylvania, 1996
- [38] Living Worlds, 1997 [http //www livingworlds com/draft.1/index htm](http://www.livingworlds.com/draft.1/index.htm)
- [39] P Maes, T Darrell, B Blumberg, and A Pentland The ALIVE system Full-body interaction with autonomous agents In N Magnenat Thalmann and D Thalmann, editors, *Computer Animation*, pages 11-18 IEEE Computer Society Press, Los Alamitos, CA, 1995
- [40] D Metaxas *Physics-Based Deformable Models Applications to Computer Vision, Graphics and Medical Imaging* Kluwer, Boston, MA, 1996
- [41] M Moore, C Geib, and B Reich Planning and terrain reasoning Technical Report MS CIS-94-63, CIS, University of Pennsylvania, Philadelphia, PA, 1994

- [42] T Noma and N Badler A virtual human presenter In *IJCAI '97 Workshop on Animated Interface Agents*, Nagoya, Japan, 1997
- [43] Open Community, 1997 <http://www.merl.com/opencom/opencom.htm>
- [44] F Parke and K Waters *Computer Facial Animation* A K Peters, Wellesly, MA, 1996
- [45] K Perlin Real time responsive animation with personality *IEEE Trans on Visualization and Computer Graphics*, 1(1) 5-15, 1995
- [46] K Perlin and A Goldberg Improv A system for scripting interactive actors in virtual worlds In *ACM Computer Graphics, Annual Conf Series*, pages 205-216, 1996
- [47] D Pratt, P Barham, J Locke, M Zyda, B Eastman, T Moore, K Biggers, R. Douglas, S Jacobsen, M Hollick, J Granieri, H Ko, and N Badler Insertion of an articulated human into a networked virtual environment In *Proc of the Conf on AI, Simulation and Planning in High Autonomy Systems* University of Florida, Gainesville, 1994
- [48] B Reich *An architecture for behavioral locomotion* PhD thesis, CIS, University of Pennsylvania, 1997
- [49] B Reich, H Ko, W Becket, and N Badler Terrain reasoning for human locomotion In *Proc Computer Animation '94*, pages 996-1005 IEEE Computer Society Press, 1994
- [50] B Robertson Best behaviors Digital magic *Computer Graphics World (Supplement)*, pages S12-S19, 1996
- [51] C Rose, B Guenter, B Bodenheimer, and M Cohen Efficient generation of motion transitions using spacetime constraints In *ACM Computer Graphics, Annual Conf Series*, pages 147-154, 1996
- [52] D Rousseau and B Hayes-Roth Personality in synthetic agents Technical Report KSL-96-21, Stanford Knowledge Systems Laboratory, 1996
- [53] T Sederberg and S Parry Free-form deformation of solid geometric models *ACM Computer Graphics*, 20(4) 151-160, 1986
- [54] S Semwal, R Hightower, and S Stansfield Mapping algorithms for real-time control of an avatar using eight sensors *Presence*, 7(1) 1-21, 1998
- [55] S Stansfield Distributed virtual reality simulation system for situational training *Presence*, 3(4) 360-366, 1994

- [56] S Stansfield, D Carlson, R Hightower, and A Sobel A prototype VR system for training medics In *MMVR Proc* , 1997
- [57] B -J Ting *Real time human model design* PhD thesis, CIS, University of Pennsylvania, 1998
- [58] D Tolani and N Badler Real-time inverse kinematics for the human arm *Presence*, 5(4) 393-401, 1996
- [59] Universal Avatars, 1996 [http //www chaco com/avatar](http://www.chaco.com/avatar)
- [60] B Webber, N Badler, B Di Eugenio, C Geib, L Levison, and M Moore Instructions, intentions and expectations *Artificial Intelligence J* , 73 253-269, 1995
- [61] J Zhao and N Badler Inverse kinematics positioning using nonlinear programming for highly articulated figures *ACM Transactions on Graphics*, 13(4) 313-336, 1994
- [62] X Zhao *Kinematic control of human postures for task simulation* PhD thesis, CIS, University of Pennsylvania, 1996



Hierarchical Virtual Characters

Armin Brubertin
armin@epimagoworks.com

EPIMAGO WORKS
ANIMATION SOFTWARE

Contents

1. Introduction

→ see paper "Hierarchical Virtual Characters";

2. Asynchronous, Hierarchical Agents

→ see paper "Hierarchical Agents Interface for Animation";

3. Animation of Human Locomotion

→ see paper "Knowledge-Driven, Interactive Animation of Human Planning";

4. Motion Signal Processing

⇒ motion multiresolution filtering

→ see paper "Motion Signal Processing";

⇒ emotion from motion

→ see paper "Emotion from Motion";

5. Conclusions and Future Work

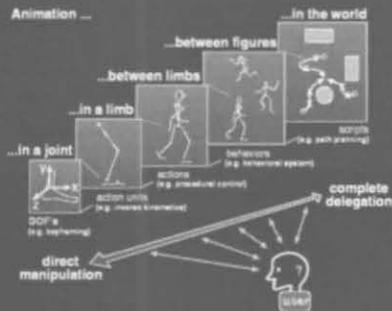
Introduction

- ⇒ Animation = creative, artistic, off-line process;
(animator needs to be in control)
 - ⇒ Simulation = scientist, verification process;
(physical laws are in control)
 - ⇒ Games = interactive, real-time process;
(user is in control)
 - ⇒ Visualization = demonstration, illustration, presentation process;
(narrator is in control)
 - ⇒ Virtual Reality = exploration, navigation process;
(real user in virtual world)
 - ⇒ Augmented Reality = enhancement, superposition process;
(virtual character in real world)
- tools are different or used differently depending on application area.

What kinds of tools do we need?

- ⇒ interactive, real-time control;
- ⇒ produce believable elementary motions:
 - a movement is believable if it is consistent with
 - the goal;
 - the internal constraints of the body;
 - constraints of biological motor control;
 - external constraints of the world.
 - Newtonian or Cartoon physics (see next slide).
- ⇒ customize and personalize elementary motions;
- ⇒ blend and overlap these motions convincingly;
- ⇒ find the right sequence of elementary motions to express behaviors.

Hierarchy of Motion Control and Specification



Contents

✓ 1. Introduction

• see paper "Hierarchical Virtual Characters";

2. Asynchronous, Hierarchical Agents

• see paper "Hierarchical Agent Interface for Animation";

3. Animation of Human Locomotion

• see paper "Knowledge-Driven, Interactive Animation of Human Running";

4. Motion Signal Processing

⇒ motion multiresolution filtering

• see paper "Motion Signal Processing";

⇒ emotion from motion

• see paper "Emotion from Motion";

5. Conclusions and Future Work

Asynchronous, Hierarchical Agent (AHA)

"autonomous computer program which performs some task or tasks on behalf of an entity, communicates asynchronously with other entities in the world and responds asynchronously to activities which occur in the agent's hierarchical internal world or the external world."

⇒ internal knowledge structure is hierarchical;

⇒ user interaction at several layers;

⇒ asynchronous agent communication:

- between internal layers;
- between agents and world;

Agent Architecture

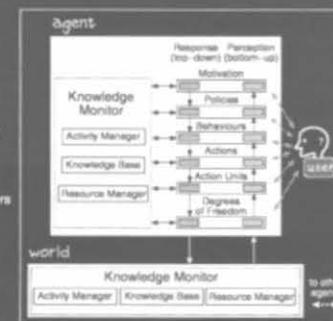
⇒ hierarchy of conceptual levels:

- motivations (goals, basic needs);
- policies determine which behavior(s) to be activated;
- behaviors are goal-directed (sequences of actions);
- actions are automated sequences of atomic action units;
- action units are sets of degrees of freedom;
- degrees of freedom are effectors and sensors which directly interact with the world;

⇒ each level only communicates with its neighboring level;

⇒ each level may communicate with the Knowledge Monitor;

⇒ each layer has Response and Perception functions;



Knowledge Monitors

↔ local to each agent + one global world.

1. Knowledge Base states of agents and facts about world.

2. Resource Manager allocation of internal agent and world resources.

3. Activity Manager

↔ manages asynchrony in agents and world.

↔ keeps track of

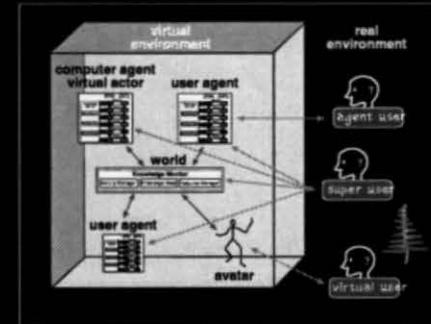
- activities running concurrently (activity list) at multiple levels;
- events (starting and stopping activities);
- filters.

{start} {running expression} {stop} → command to call

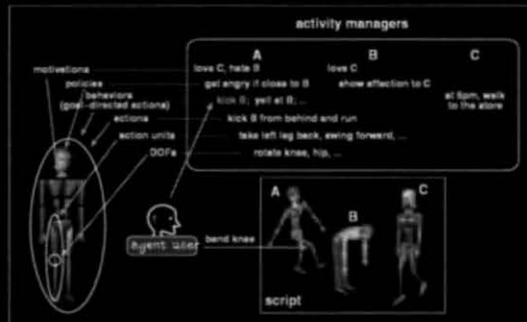
examples:

```
{raining} {outside & have_umbrella} {} → "start use_umbrella"
{} {use_umbrella} {raining} → "stop use_umbrella"
{close to B} {walking & afraid_of_B} {} → "start fast_walking"
```

Agent Interface



Agent Example



Summary

- ↔ framework for asynchronous, hierarchical multi-agents can be applied to create human agents (virtual actors and user agents);
- ↔ top-down (response) and bottom-up (perception) multi-layered approach;
- ↔ application examples:
 - malefiz game;
 - simulation of a crowd scene;
 - director's assistant;
- ↔ goals satisfied:
 - ✓ - interactive, real-time control;
 - produce believable elementary motions;
 - customize and personalize elementary motions;
 - blend and overlap these motions convincingly;
 - ✓ - find the right sequence of elementary motions to express behaviors.

Contents

✓ 1. Introduction

➤ see paper "Hierarchical Virtual Characters";

✓ 2. Asynchronous, Hierarchical Agents

➤ see paper "Hierarchical Agent Interface for Animation";

3. Animation of Human Locomotion

➤ see paper "Knowledge-Driven, Interactive Animation of Human Running";

4. Motion Signal Processing

⇒ motion multiresolution filtering

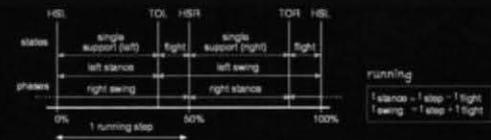
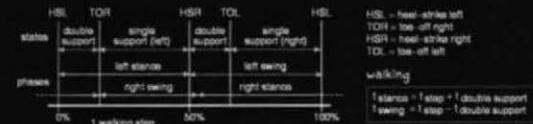
➤ see paper "Motion Signal Processing";

⇒ emotion from motion

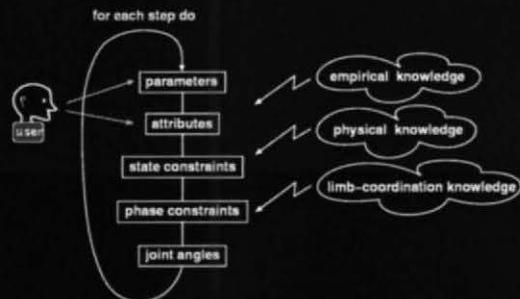
➤ see paper "Emotion from Motion";

5. Conclusions and Future Work

Locomotion Cycle



Locomotion Algorithm



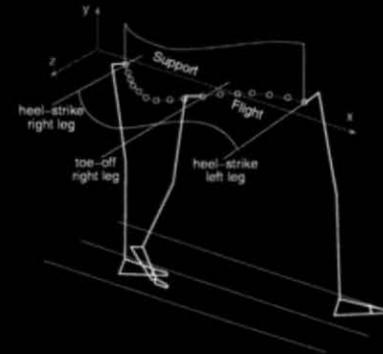
Running Parameters

- define locomotion stride;
- interactively adjustable via sliders;
- 4 running parameters:
 - ⇒ velocity (v);
 - ⇒ step length (sl);
 - ⇒ step frequency (sf);
 - ⇒ flight height (H).
- relationship determined by empirical knowledge and physical constraints.

Running Attributes

- finetune (define style of) locomotion stride;
- interactively adjustable via sliders;
- currently 19 running attributes, e.g.
 - ⇒ amount of arm swing, pelvic rotation, bounciness heel/toe-strike, stride width;
- relationship determined by empirical knowledge and physical constraints;
- dependent on parameters, e.g.
 - ⇒ pelvic rotation increases with step length;

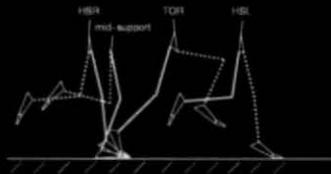
State Constraints



- define the motion of body during a step (dotted curve);
- define "internal" keyframes for the legs at heel-strike and toe-off based on the parameter and attribute values for the current step.

Phase Constraints

- subdivide the "keyframes" generated by the state constraints;
- ⇒ for instance, for the swing leg (dashed leg in the slide) we introduce two new keyframes at mid-support and toe-off.
 - at mid-support, the thigh is vertical and the knee angle reaches maximum flexion;
 - at toe-off, the hip is maximally extended and the knee angle is such that the toe is directly vertically under the knee.



Results

various walking styles



various running styles



Summary

- ⇒ system generates a wide variety of human walks and runs;
- ⇒ 80 angles for 37 joints are calculated;
- ⇒ goals satisfied:
 - ✓ - interactive, real-time control;
 - ✓ - produce believable elementary motion;
 - ✓ - customize and personalize elementary motions;
 - find the right sequence of elementary motions to express behaviors.
 - blend and overlap these motions convincingly;
 - find the right sequence of elementary motions to express behaviors.

Contents

✓ 1. Introduction

- see paper "Hierarchical Virtual Characters";

✓ 2. Asynchronous, Hierarchical Agents

- see paper "Hierarchical Agent Interface for Animation";

✓ 3. Animation of Human Locomotion

- see paper "Knowledge-Driven, Interactive Animation of Human Running";

4. Motion Signal Processing

⇒ motion multiresolution filtering

- see paper "Motion Signal Processing";

⇒ emotion from motion

- see paper "Emotion from Motion";

5. Conclusions and Future Work

Motion Signal Processing

what?

- apply image and signal processing techniques to reuse, modify and adapt animated motion of figures with many degrees of freedom.

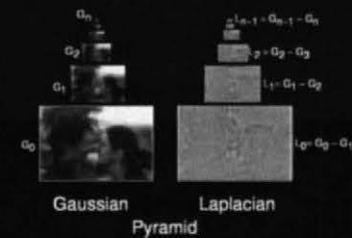
why?

- provide higher-level motion editing at interactive speeds.
 - make libraries of animated motion more valuable.



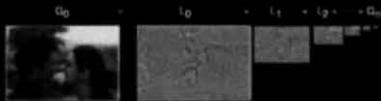
Multiresolution Filtering

Decomposition:



Multiresolution Filtering

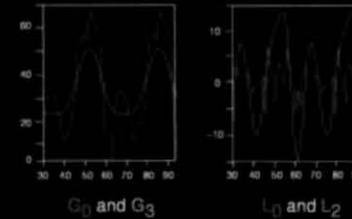
Reconstruction:



examples:
- image merging
- temporal image dissolves

Motion Multiresolution Filtering

example: knee angle of a walking sequence



Motion Multiresolution Filtering

⇒ Algorithm 1: decomposition and reconstruction of motion parameters.

for each motion parameter G_0 of length m

1. $G_{k+1} = w_{k+1} \cdot G_k$ for all k , $0 \leq k < fb$, where $fb = \log_2(m)$.
2. $L_k = G_k - G_{k+1}$.
3. $G_0 = G_{fb} + \sum_{k=0}^{fb-1} L_k$

Motion Multiresolution Filtering

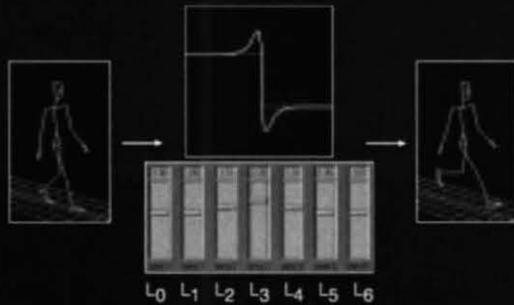
⇒ Algorithm 2: decomposition and construction of motion parameters by adjusting frequency band gains.

for each motion parameter G_0 of length m

1. $G_{k+1} = w_{k+1} \cdot G_k$ for all k , $0 \leq k < fb$, where $fb = \log_2(m)$.
2. $L_k = G_k - G_{k+1}$.
3. adjust gains g_k for each L_k .
4. $G_0 = G_{fb} + \sum_{k=0}^{fb-1} (g_k \cdot L_k)$

examples:
- toning-down motion
(increase low frequency gains)
- exaggerating motion
(increase middle frequency gains)
- nervous movement
(increase high frequency gains)
- smoothing motion
(reduce high frequency gains)

Motion Multiresolution Filtering



Emotion from Motion

what?

- generate "emotional transform" and apply to "neutral" movement to make it emotional.

an animated sequence of a person knocking at a door

"angry" transform

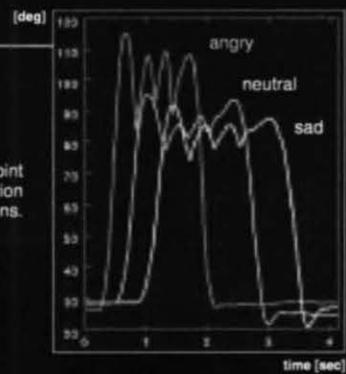
an angry person knocking

why?

- make libraries of animated motions (keyframed, motion-captured, simulated) more valuable.

Example

motion captured elbow joint angle of a knocking motion with different emotions.



Emotion from Motion

1. derive emotional transform ("difference" between neutral and emotional movement):

⇒ speed transform (dynamic timewarping):

- divide movement into basic periods;
- timewarp periods from neutral to emotional movement.

⇒ amplitude transform (signal amplifying):

- divide joints into categories;
- for each category and each basic period, extract spatial amplitude from neutral and emotional movement.

⇒ [frequency transform.]

2. apply emotional transform (to new neutral movement):

⇒ calculate basic periods of new neutral movement.

⇒ apply speed and amplitude transforms to each basic period.

Speed Transform

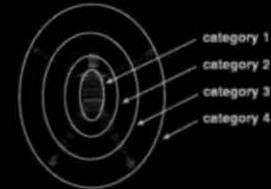
- basic periods:



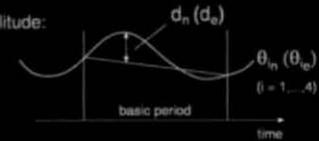
- timewarp periods from neutral to emotional movement.

AmplitudeTransform

- categories:

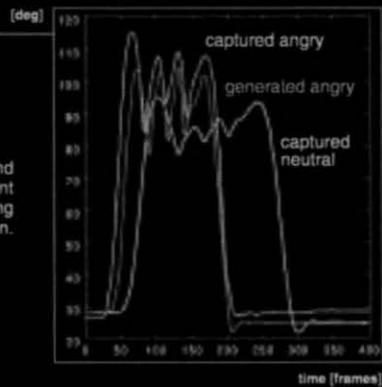


- spatial amplitude:



Result

motion captured and generated elbow joint angles of a knocking motion.



Summary

- ⇒ facilitate the reuse and adaptation of motion data (motion capture and libraries);
- ⇒ provide rapid interactive loop;
- ⇒ lend themselves to high level motion control;
- ⇒ support operations like blending, concatenation of motions; capping joint angles; tone-down, exaggerate motion;
- ⇒ apply styles (emotions) to monotonic (neutral) motion;
- ⇒ displacement mapping provides "keyframe" interface to edit motion captured data.
- ⇒ goals satisfied:
 - ✓ - interactive, real-time control;
 - produce believable elementary motion;
 - ✓ - customize and personalize elementary motions;
 - ✓ - blend and overlap these motions convincingly;
 - find the right sequence of elementary motions to express behaviors.

Contents

✓1. Introduction

➤ see paper "Hierarchical Virtual Characters";

✓2. Asynchronous, Hierarchical Agents

➤ see paper "Hierarchical Agent Interface for Animation";

✓3. Animation of Human Locomotion

➤ see paper "Knowledge-Driven, Interactive Animation of Human Running";

✓4. Motion Signal Processing

⇒ motion multiresolution filtering

➤ see paper "Motion Signal Processing";

⇒ emotion from motion

➤ see paper "Emotion from Motion";

5. Conclusions and Future Work

Conclusions, Future Work

⇒ hierarchical motion control framework for multiple virtual characters with

- multi-level interface;
- asynchronous management of activities;
- top-down (response) and bottom-up (perception).

⇒ we are currently developing building blocks:

- interactive, real-time control;
- tools to create believable elementary motions (walking, running, etc.);
- customize and personalize elementary motions;
- tools for blending and overlapping these elementary motions;
- tools for producing "emotional" from "neutral" motion.

⇒ application examples:

- video games;
- personal information and interface agents for virtual environments;
- animation: e.g. simulation of a crowd scene; director's assistant.

Hierarchical Virtual Characters

Armin Bruderlin
Sony Pictures Imageworks
Culver City, CA 90232
email armin@spimageworks.com

1 Introduction

This talk focuses on controlling and interacting with virtual characters. Just as real humans can perform a rich set of behaviors and actions in the real world, we would like virtual humans to be able to perform similar tasks in a virtual environment upon “convenient” specification by the user.

In the previous presentation by Norman Badler, a wide variety of issues relating to this topic have been discussed. Badler has presented a well-organized overview, addressed various application areas, research topics and solutions to many problems, including task planning, multi-agent interaction and language-based control. In this talk, we present a much smaller, not-all-encompassing framework for *multi-level* control of and interaction with virtual characters. We argue that a hierarchical structure for motion generation, specification and modification lends itself well to control the movements of complex virtual characters.

It is clear that the requirements and tools for animating virtual characters vary across different application domains. For instance, traditional *Animation* is a creative, artistic, off-line process where animators often prefer to have fine-control to tune all the parameters. *Simulation* is a scientific, verification process, *Video Game* playing is an interactive, real-time process, *Visualization* is a demonstration and illustration process, *Virtual Reality* is an exploration and navigation process (real user in virtual world), *Augmented Reality* is an enhancement and superposition process (virtual character in real world). Nonetheless, we hope that the following requirements can serve as guidelines to develop useful tools across disciplines.

- provide interactive, real-time control,
- produce *believable* elementary motion
- customize and personalize elementary motions,
- blend and overlap these motions convincingly,
- find the right sequence of elementary motions to express behaviors

All the tools introduced in this talk provide interactive, real-time control in that the system responds immediately to a change of a motion parameter by the user.

The tools also satisfy one or more of the remaining requirements as described in section 3 below.

We use the term “believable” to mean a movement which is consistent with the goal, the internal constraints of the body, the constraints of the motor control system, and the external constraints of the world. In this way, we do not constrain a motion to be “realistic” in a sense that it has to exactly follow Newton’s laws of motion, but may instead be based on *cartoon* physics. This is an important feature of the virtual world as a medium, which goes beyond its real counterpart: as long as the rules are well-defined, consistent and comprehensive to the user or audience, virtual characters should be allowed to exaggerate, squash and stretch, etc., in some cases to convey messages more clearly.

The generation of elementary motions can be achieved in a variety of ways, including the use of motion capture, keyframing and simulation. The requirement for tools to facilitate customizing and personalizing elementary motions as well as blending and overlapping these motions becomes especially valuable for reusing motion captured data and building up more complex movement sequences. Finally, tools for finding the right sequence of elementary motions to express higher-level behaviors are useful when interacting with virtual characters in games or virtual reality applications.

2 Multi-level Motion Control and Specification

We think that the above requirements can be best realized in a hierarchical motion control scheme which provides a layered interface for mappings between higher-level control parameters (e.g. desired walking speed for locomotion) and actual low-level motion parameters (e.g. joint angles). Of course, the number of control parameters is usually much smaller and more meaningful than the number of motion parameters. The term “convenient” in the introduction thus refers to the fact that a user can interact with the system at multiple levels of abstraction, from very low-level, *in a joint*, direct manipulation (e.g. keyframing) to very high-level, delegation mode. This is illustrated in Figure 1.

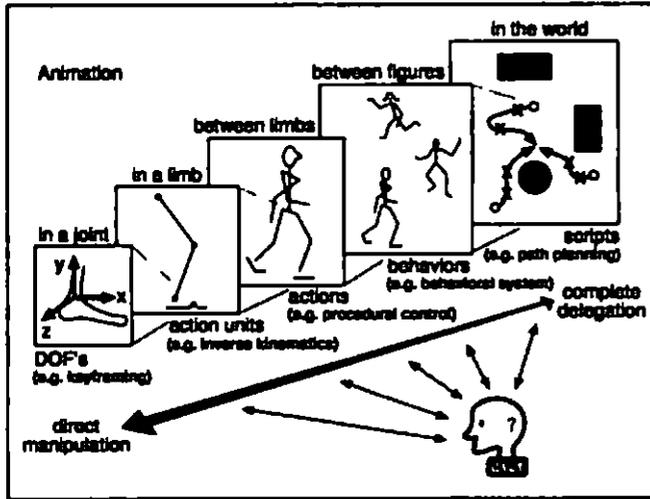


Figure 1 Virtual character hierarchy for motion control and interaction

3 Organization of Talk

Three topics are addressed in this talk. They all focus around the motion control hierarchy in Figure 1. Each topic is accompanied by video clips and corresponding papers which are included in these course notes.

- Asynchronous, Hierarchical Agents
- Animation of Human Locomotion,
- Motion Signal Processing

The asynchronous hierarchical agent system allows for a layered definition of virtual agents according to Figure 1. The agents can interact asynchronously with each other, and respond to user interactions at the different levels of abstraction depending on their needs. Although we currently have only very simple agents executing elementary motions, this approach promises to be extended to more complex human-like characters. With respect to our list of requirement for motion control tools, the asynchronous, hierarchical agents can be used to find the right sequence of elementary motions to express desired behaviors. For more detail on this approach, read the enclosed paper "Hierarchical Agent Interface for Animation" [3].

The second topic describes our interactive real-time animation system for human locomotion. Within the hierarchy of Figure 1, we provide control at the "between limbs" level, where the user can interactively adjust certain locomotion parameters and attributes like speed, step length, pelvic rotation or amount of arm swing to customize the walk or run of a virtual character which is being displayed at the same time. The locomotion system satisfies the following requirements from above: it produces believable motion, and it allows for customizing and personalizing locomotion. The paper in these course notes entitled "Knowledge-Driven,

Interactive Animation of Human Running" [2] has more detail.

Finally, we present motion signal processing tools as a means to customize and personalize elementary motions, as well as blend and overlap these motions convincingly. In reference to the layers in Figure 1, these tools provide "between limbs", "in a limb" or "in a joint" editing control. Two papers are included on this topic: "Motion Signal Processing" [4] and "Emotion from Motion" [1]. Whereas the former describes low-level tools like motion multiresolution filtering and motion displacement mapping for interactive adaptation of existing elementary motions, the latter paper shows an approach to automatically generate emotional movement from neutral motion.

References

- [1] AMAYA, K, BRUDERLIN, A, AND CALVERT, T. Emotion from motion. In *Graphics Interface '96, Proceedings* (May 1996) pp 222-229.
- [2] BRUDERLIN, A, AND CALVERT, T. Knowledge-driven, interactive animation of human running. In *Graphics Interface '96, Proceedings* (May 1996), pp 213-221.
- [3] BRUDERLIN, A, FELS, S, ESSER, S, AND MASE, K. Hierarchical agent interface for animation. To appear in *IJCAI'97, Workshop on Animated Interface Agents* (August 1997).
- [4] BRUDERLIN, A, AND WILLIAMS, L. Motion signal processing. In *Computer Graphics (SIGGRAPH '95 Proceedings, Los Angeles CA August 6-11)* (1995), pp 97-104.

Hierarchical Agent Interface for Animation

Armin Bruderlin and Sidney Fels and Silvio Esser and Kenji Mase

ATR Media Integration & Communication

Seika-cho Soraku-gun

Kyoto 619-02, Japan

email armin,fels,esser,mase@mic atr co jp

Abstract

Asynchronous, Hierarchical Agents (AHAs) provide a vertically structured multilevel abstraction hierarchy. In this paper, we argue that this multilevel hierarchy is a convenient way to create a human-agent interface at multiple levels of abstraction. In this way, the agent has several layers of *specification* (input) and *visualization* (output) which facilitates users with problem solving, because such an interface parallels the hierarchical and iterative nature of human creative thought processes. The AHA interface presents an intuitive, intimate interface which supports interactions on a scale from direct manipulation to delegation, depending on the user's choice. Another feature of this interface is its two modes of interaction: direct device interaction (mouse clicking) and interpretive, command line or scripting mode. This way, agents can be "forced" to perform certain activities via mouse clicks (direct control), or they can be programmed via scripts on the fly. We present examples of the use of the AHA interface from the domain of animating human-like agents in a virtual world.

Keywords intelligent agents, creative process, motion control

1 Introduction

In this paper, "agent" refers to an autonomous computer program which performs some task or tasks on behalf of an entity, communicates asynchronously with other entities in the world and responds asynchronously to events in the world. Entities may be people, other agents, other traditional programs or objects in the world. Our agents are structured hierarchically and respond asynchronously to events and are called Asynchronous, Hierarchical Agents (AHAs).

The term "autonomous computer program" implies that the program can run on any machine without intervention from a person. The "asynchronous" nature of the agent implies that it responds to events which occur as-they-happen rather than requiring a global synchronization signal. Of course, this does not mean that synchronous behaviour cannot be implemented by asynchronous agents, rather, synchronization must be made explicit where it is needed.

The term "hierarchical" means that the internal structure and knowledge of the agent is organized in several conceptual layers. These layers also correspond to the levels of interaction which the user has to communicate with such an agent. Our AHAs use a vertically layered approach [Müller *et al*, 1994]. The main focus of this paper is to discuss the hierarchy with respect to the different levels and modes of interaction. The combination of multiple levels, which provide access to the activities of agents from low-level, direct manipulation to high-level, complete delegation, together with mouse-click and scripted interaction modes results in a flexible interface which represents a useful and convenient approach to real-time interaction with virtual characters in virtual worlds.

In the next section, related work is described. Section 3 explains the asynchronous agent hierarchy in more detail, and introduces the different modes of interaction within this hierarchy. Examples from animation of virtual worlds are given in section 4. Finally, section 5 gives conclusions and outlines future work.

2 Related Work

Our work on hierarchical agents continues the tradition of layered computation [Brooks, 1986] [Ferguson, 1992] [Bonasso *et al*, 1995]. The combination of a layered approach and a belief, desire, and intention (BDI) architecture found in the InteRRaP system [Fischer *et al*, 1995] [Müller *et al*, 1994] most closely resembles our approach. Like [Fischer *et al*, 1995], we take a pragmatic approach to the problem of designing an agent and do not propose a new theory for knowledge representation. In their model, the knowledge base is structured hierarchically with increasing knowledge abstraction. Likewise, agent control is layered functionally with the behaviour-based layer (BBL), the local planning layer (LPL) and the cooperative planning layer (CPL). Access from the control layers to the knowledge base maintains the hierarchy abstraction. Further, in the InteRRaP system, there is a world interface unit (WIF) which allows information flow to and from the world. In our system, we have a structure called the knowledge monitor which contains a knowledge base that is structured so that all control layers have access to the information contained, providing

access to knowledge that applies to all levels of control abstraction. Our hierarchical decomposition of knowledge is also similar in spirit to [Blumberg and Galyean, 1995], who propose a three-layer system to control the behaviours of Silas a virtual dog.

One of the motivations for our design is to allow the agentee (agent user) to have control over the agent at different levels of abstraction. The philosophy is related to Norman's seven steps of human action [Norman 1988]. The 7 steps are generally divided into goal, intention, action sequence, execution, perception, interpretation, evaluation. While we divide our agent's actions into different layers (see section 3.1 below), we are consistent in considering that the agent acts upon the world (and itself) and evaluates changes in the world (and itself) as a result.

3 AHA System

A block diagram of an Asynchronous Hierarchical Agent (AHA) is shown in Figure 1. The three fundamental aspects in the system are

- 1 hierarchical agent structure,
- 2 knowledge monitor,
- 3 user interaction.

The remainder of this section describes each of these parts in detail.

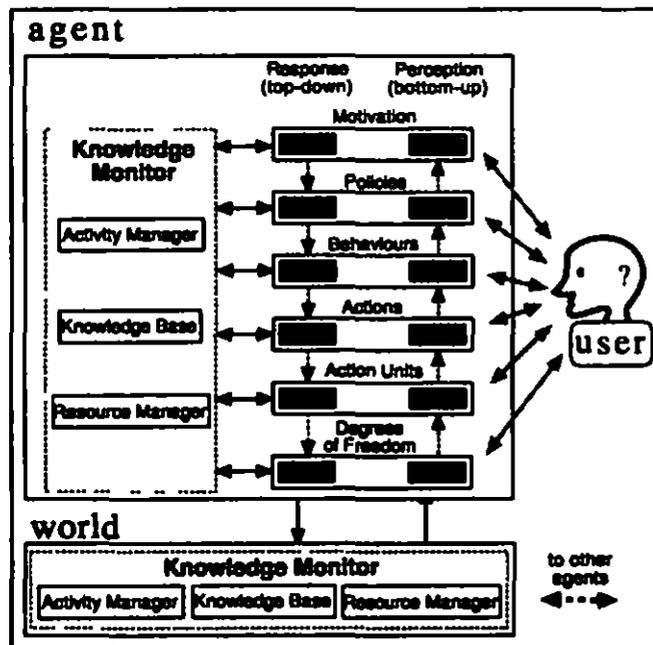


Figure 1 AHA structure

3.1 Hierarchical Agent Structure

The multi-level hierarchical structure of the system is comprised of the following layers

- 1 Motivation (needs)
- 2 Policy (plan)
- 3 Behaviour (goal-directed [sequences of] actions)
- 4 Action (automated sequences of atomic action units),
- 5 Action Units (sets of degrees of freedom)
- 6 Degrees of Freedom (effectors and sensors which interact with the world)

Each layer corresponds to a separate level of functional and conceptual activity inside the agent and communicates (through the knowledge monitor) to its neighbours. Typically, for responding to events which have occurred in the world, information flows from the top of the hierarchy to the bottom eventually reaching the bottom layer which can act on the world directly. When the agent is trying to understand and perceive the world, information flows from the bottom of the hierarchy to the top. The dotted connections indicate logical data-flow. All real data flow occurs through the knowledge monitor.

3.2 Knowledge Monitor

The second component is called the *knowledge monitor* and is made up of three parts: the *knowledge base* which contains global variables and knowledge shared between layers, the *resource manager* which administers the co-ordination of resources, the *activity manager*.

The activity manager is implemented using a client/server model. The server keeps track of three main types of information specified by clients (layers in hierarchy), first, an activity list containing all the activities which are concurrently happening, second, events which occur (i.e. starting or stopping of activities), third, a list of all the perceptual filters watching for events and activities to occur. A filter is specified by a client as a four-tuple of the following form:

```
{ start      {running activity  { stop
 activity}   expression}      activity}
                                     --> {command to execute}
```

When *all* the conditions specified in the filter are true then the filter triggers the command associated with it. Here are two examples:

```
{raining} {outside & have_umbrella} { }
                                     --> "start use_umbrella"

{ }      {use_umbrella}      {raining}
                                     --> "stop use_umbrella"
```

As seen in Figure 1, the knowledge monitor structure is also used to create a knowledge monitor for the world. This knowledge monitor is used *stand-alone* to maintain the world state and keep track of events which occur in it.

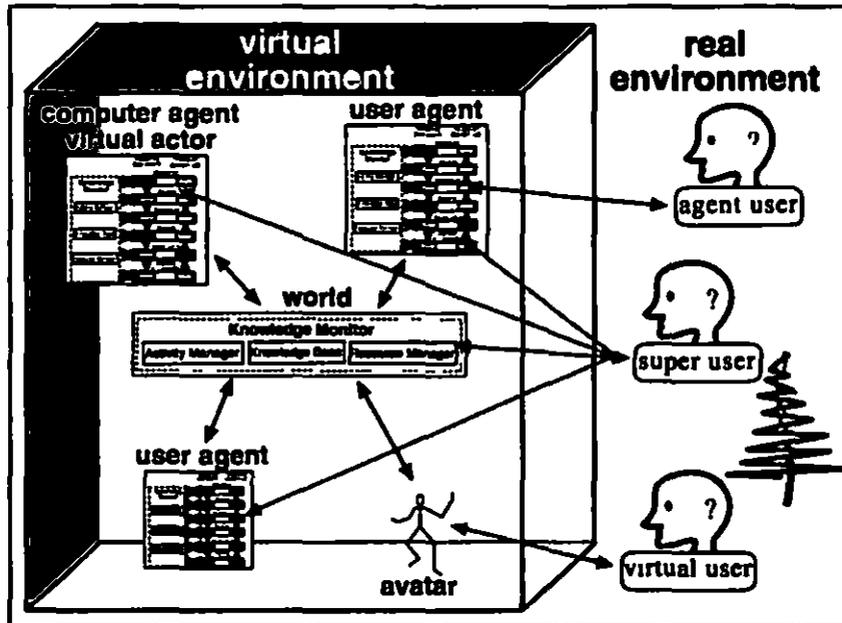


Figure 2 Interactions between users and virtual environment

3.3 User Interaction

When agents interact with an agent or virtual character (see Figure 1), the different layers of abstraction provide a convenient decomposition related to their own way of thinking. Abstraction is an important concept for designing an agent interaction process: it allows for the convenient creation and modification of the agent by the agentee at different levels of abstractions or granularities, enabling the viewing of a "tree" without knowing about "forests", and the examination of a "forest" without distraction of the details of the "trees". We can consider the control of virtual characters¹ a complex synthesis task similar to the composition of a musical piece or the design of a new product. Such creative processes are inherently hierarchical, iterative, and make use of alternate views of the problem to find a solution [Simon, 1969]. Koestler notes that the act of discovery or creation often occurs when distinct representations are recognized as depicting the same object, idea, or entity [Koestler, 1990]. Thus, by having both, levels of knowledge and levels of interaction represented in this hierarchical framework, users can choose the level of detail with which to specify and visualize at any moment, depending on their current state of mind.

A more complete illustration of interaction with agents in a virtual environment is shown in Figure 2. Three types of users can be distinguished: the *agent user*, who is represented in the virtual world by his/her *user agent*, i.e. an Asynchronous, Hierarchical Agent as discussed in the last section, which can autonomously carry out tasks or be controlled at multiple layers of abstraction,

the *super user*, who can access anything in the virtual world, e.g., he can initialize the world, user agents, as well as *virtual actors*, which are agents without agent users. Finally, there is the *virtual user*, who has no agent support, but is represented by an *avatar* whose actions are completely controlled by its virtual user at any time. We are currently working on implementing such a system based on the AHA architecture. Examples are given in the next section.

4 Examples

Figure 3 shows an example of an agent user interacting with his user agent A via the multi-layered interface. Agent A has chosen the indicated motivations, policies, behaviours, etc. and its agent user has decided to make A "kick B" this time by interactively setting A's behaviour. Likewise, an agent user might choose low-level, direct manipulation control by bending the knee of his agent a bit more.

Currently, our AHA hierarchical agent/interface project is still at a developing stage, and we do not yet have a general system to control human-like agents as illustrated in Figure 3. However, we have implemented the basic AHA framework, and are now testing it in the simplified scenario of a board game, called the *Malefiz* game [Esser *et al.*, 1997]. Figure 4 shows a snapshot of the game in progression. There are four players (with five pieces each), each implemented as an Asynchronous Hierarchical Agent. The players know the rules of the game and act autonomously or with respective agent user interactions with the layers. For instance, player A might have the motivation "winning the game" which triggers the policy "aggressive" which in turn activates the behaviour "kick out player B" (after the proper die roll).

¹This holds, in general, also for any complex computer program, or a program in a complex environment.

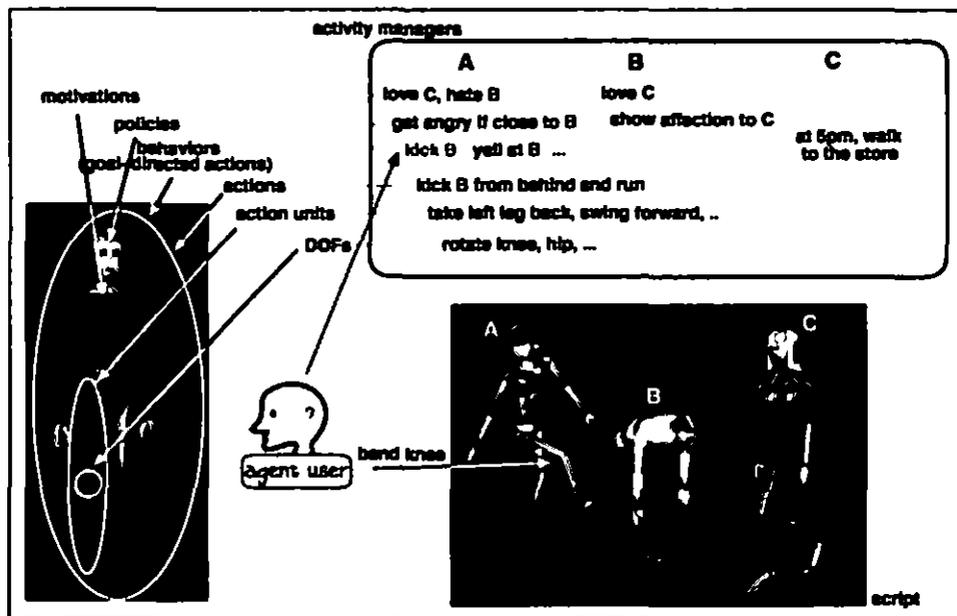


Figure 3 Multi level interaction via scripts and direct manipulation

which will result in the two actions of player B being sent home and player A being positioned where player B was. However, the user agent for A might decide instead not to kick out B but advance through a different path, by overwriting the policy "aggressive" with "advancing". There might also be a virtual user participating in the game i.e. a user without agent support who has to move his/her pieces "by hand" with a set of proper move commands



Figure 4 Snapshot of AHA Malefiz game

Another current research effort towards a general AHA system to control human-like agents in virtual environments is focusing on implementing parameterized low-level, believable behaviours, elementary mo-

tions (like kicking, walking, running, knocking), as well as tools for customizing and personalizing blending and overlapping these elementary motions. Figure 5 shows an example of our interactive human locomotion system [Bruderlin and Calvert 1996], where the agent user can interactively specify certain parameters like speed, step length, amount of arm swing, torso tilt while the walking or running human-like agent reflects these changes. This is an example of direct manipulation by dragging sliders to affect changes in motion. We are currently looking into "hooking up" the walking and running algorithms into the AHA structure, so that these movements can be called via scripts and the personality or style of the motion (i.e. the value of the sliders) is set by the motivation and policy layers.

5 Conclusions and Future Work

We have introduced a framework for asynchronous hierarchical agents and discussed its relevance from the perspective of multi-layered human agent interfaces. We believe that such an interface supporting multiple levels of abstraction facilitates interactions with agents in a virtual environment, as users can choose their level of involvement at any time to suit their thinking.

After successful demonstration of the basic principles of the AHA architecture within the Malefiz game, we are looking into providing a more complex example, such as animating a crowd scene, and creating personal information and navigation agents for visitors to our Meta-Museum project. We also want to incorporate learning mechanisms into our hierarchy, so that agents will improve or adapt their actions over time based on user input and the actions of other agents.

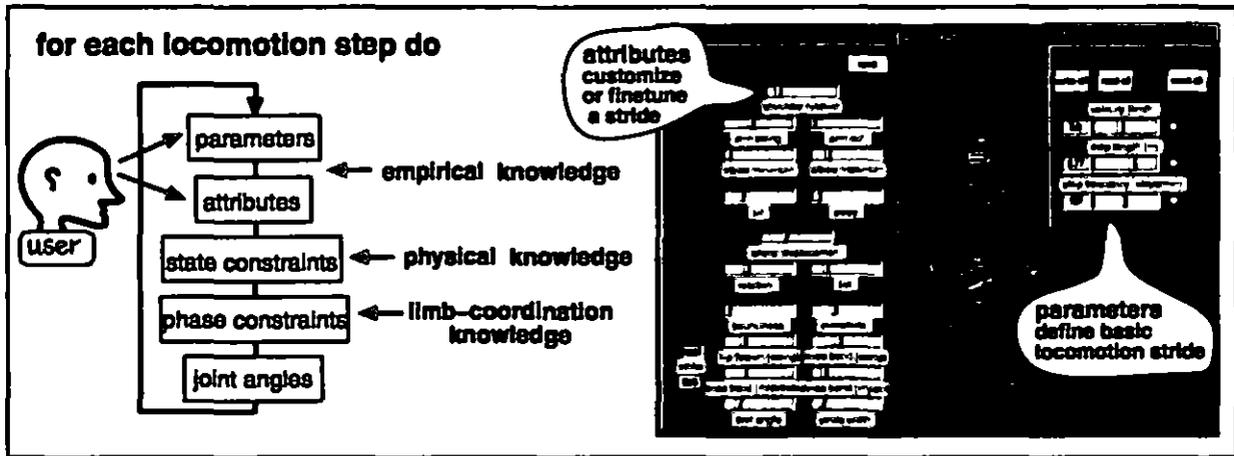


Figure 5 Direct manipulation of parameters for human locomotion

References

- [Blumberg and Galyean, 1995] B M Blumberg and T A Galyean Multi-level direction of autonomous creatures for real-time virtual environments In *SIGGRAPH 95*, pages 47-54, August 1995
- [Bonasso *et al*, 1995] R. P Bonasso, D Kortenkamp, D P Miller, and M Slack Experiences with an architecture for intelligent reactive agents In M Woolridge, J P Muller, and M Tambe, editors, *Intelligent Agents II*, pages 187-202 Springer, 1995
- [Brooks 1986] R. A Brooks A robust layered control system for a mobile robot *IEEE Journal of Robotics and Automation*, 2(1) 14-23, April 1986
- [Bruderlin and Calvert, 1996] Armin Bruderlin and Tom Calvert Knowledge-driven, interactive animation of human running In *Graphics Interface '96, Proceedings*, pages 213-221, May 1996
- [Esser *et al*, 1997] S Esser, S Fels, A Bruderlin, and K Mase Inventcl A 3d mega-widget using open inventor and tcl/tk submitted for publication in Usenix, Tcl/Tk workshop, 1997
- [Ferguson, 1992] I A Ferguson *TuringMachines An Architecture for Dynamic, Rational, Mobile Agents* PhD thesis, University of Cambridge, UK, 1992
- [Fischer *et al*, 1995] K Fischer, J P Müller, and M Pischel A pragmatic bdi architecture In M Woolridge, J P Müller, and M Tambe, editors, *Intelligent Agents II*, pages 203-218 Springer, 1995
- [Koestler, 1990] G koestler *The Act of Creation* reissued edition, Arkana, 1990
- [Müller *et al*, 1994] J P Müller, M Pischel, and M Thiel Modelling reactive behaviour in vertically layered agent architectures In M Woolridge and N R Jennings, editors, *Intelligent Agents*, pages 261-276 Springer-Verlag, 1994
- [Norman, 1988] D Norman *The Psychology of Everyday Things* Basic Books Inc New York, 1988
- [Simon, 1969] H A Simon *The Sciences of the Artificial* MIT Press, Cambridge, MA, 1969

Knowledge-Driven, Interactive Animation of Human Running

Armin Bruderlin
ATR Media Integration & Communications Research Labs
Seika-cho Soraku-gun
Kyoto 619-02, Japan
email.armin@mic.atr.co.jp

Tom Calvert
Simon Fraser University
Burnaby, B C , V5A 1S6, Canada
email.tom@cs.sfu.ca

Abstract

A high-level motion control system for the animation of human-like figures is introduced which generates a wide variety of individual running styles in real time. Sequences such as a leisurely jog, a fast sprint or a bouncy run are conveniently obtained by interactively setting the values of "running" parameters such as desired velocity, step length, "flight" height or "heel/toe" strike while observing a running figure on the screen.

The algorithm incorporates knowledge of how humans run at several levels. *empirical* knowledge defines the relationships between the running parameters, for example, a change in running velocity by the user triggers a change in step length to maintain a "natural" running stride. *Physical* knowledge calculates the trajectory of the body for the current running step. Knowledge about limb-coordination of a running stride is utilized for establishing both, *state constraints* which define the support and flight states, and *phase constraints* to "guide" the internal joint-angle interpolation for the stance and swing phases.

Keywords human figure animation, motion control, procedural animation

1 Introduction

Animating the motion of human figures is a challenging task. Traditionally, an animator has to tediously specify many keyframes for many degrees of freedom to obtain a desired movement. At the same time, spatial and temporal components of a movement, coordination of the limbs, interaction between figures as well as interaction with the environment need to be resolved. Keyframing supports the animation process only at the lowest level providing tools for the manipulation of joint angles or coordinates (i.e. low level motion parameters), and the animator has to explicitly account for higher level interactions based on experience and skills.

We propose a higher level motion control technique to animate human running which alleviates the tedious detail the animator has to specify. A higher level of control is achieved by incorporating knowledge about how people run into the motion control algorithm. High-level motion parameters such as velocity, flight height, pelvic rotation or stride width generate variations in running styles with

different expressions. These parameters can be changed interactively to give real-time feedback on their effect on the motion — analogous to 'tweaking' low-level parameters such as control points of joint angle trajectories in a keyframing system.

Several other approaches have been proposed to animate movements of human-like figures at a higher, above joint-manipulation level. *Inverse kinematics* has been successfully applied to ease limb positioning [5] and motion generation [20, 25] by encoding knowledge on how the joint angles change given the position of an end-effector. *Physically-based* approaches [14, 29] incorporate knowledge in form of dynamic models to produce realistic although often expressionless motion, which is also difficult to control since the driving forces and torques need to be approximated for a particular movement. Hybrid kinematic-dynamic techniques [18, 30] have been proposed to avoid some of the disadvantages associated with purely dynamic systems. *Optimization* techniques have been applied to simplified articulated structures to generate motions such as throwing [9, 23] and lifting [22], here, knowledge is embedded in a cost function term and constraints whose solution approximates a desired movement. Motion control at a very high level is provided by *planning* systems which have knowledge of an entire scene, and a planner autonomously generates the animation [6, 20], including gestures and conversations between characters [7]. However, such high level systems often produce plain animations lacking expression and personality.

Most higher level approaches also share the drawback of not performing in real time, making it difficult for the animator to interactively and iteratively construct and refine a desired movement. There is usually a trade-off between how much and what the animator has to specify, and how expressive and close the resulting motion is to what the animator had in mind, that is how much the system "assumes" about motion. For instance, traditional keyframing is an 'assisting' tool which assumes very little about a motion by mainly taking care of interpolation and administration of the input, while it is really the animator who does motion control. On the other hand, animating a bouncing ball in a physically-based system leaves the animator with very little control for fine tuning after the equations of motion have been set up and initialized.

The disadvantages of lack of control, lack of expression and non-interactivity usually attached to higher level systems have been overcome by our motion control algorithm for human running. The creative control stays with the animator who can interactively adjust high-level parameters to obtain a desired running style. In section 2 related research in animating human locomotion is outlined and the main concepts of human running are defined. Section 3 describes the motion control algorithm for running in detail followed by a discussion and examples in section 4. Conclusions and extensions for future work are addressed in section 5.

2 Bipedal Running

In human locomotion walking and running are the most important and most frequently used gaits. A lot of investigation has been done on human walking (see [17] for a good overview) and at least conceptually and kinematically it is well understood. On the other hand, there is not as much consensus in research on running. This has much to do with the fact that there is a larger number of running styles compared to walking. For example, humans can run at any speed at which walking is possible, they can run leisurely at a modest pace they can jog or sprint at maximum speed, which results in a much greater variability in the kinematics than, say, between a slow and a fast walk. Furthermore the choice between one of the two basic running styles—either the toe or the heel impacting with the ground first at the end of a running step—has different effects on the kinematics (and the dynamics which are ignored here) of the foot and leg during stance. Variations between running subjects are also more pronounced than in different people walking because of the flight or airborne state which does not occur during a walking stride. Lastly, we believe that anatomical differences between humans such as unequal leg lengths and muscle strengths produce a wider range of kinematic patterns in running than in walking since strength becomes more of a factor.

2.1 Related Work

Early work in animating legged figures includes PODA [12] which produces a variety of gaits. Simple dynamics are applied to calculate the motion of the body as a whole, while the legs are animated kinematically using a pseudoinverse Jacobian technique to determine the leg angles while keeping the feet on the ground during support. A different approach to animate legged locomotion was taken by Raibert et al [26]. An internal control algorithm for a dynamic running model is decomposed into three independently treated functions: a vertical component to regulate hopping height, and a horizontal component regulating body attitude (balance) and forward velocity. Hodgins [13, 14] introduced a related approach to dynamically animate human running. The control algorithm relies on a cyclic state machine which determines the proper control actions to calculate the forces and torques such that the desired forward speed is satisfied. Stewart [27] presented another active control dynamically based system which allows the user to write an algorithm (in LISP) for a particular motion.

The equations of motion are constructed by the algorithm which then controls the motion by setting values of variables and keeping track of the state of the simulation. Although convincing examples of walking sequences of a simple biped have been demonstrated, this approach requires the user to know a lot about the motion to be animated by writing an appropriate algorithm. In an effort to produce more "human-like" motion, techniques have been developed to generalize rotoscoped data for animating human walking [2, 19] such that variations in direction and step lengths for figures of different height can be obtained.

A different approach was taken with the development of K LAW [3, 4]. By incorporating knowledge about how humans walk, the system generates realistic human walk animations at a high level while still allowing for variations in the movement. In this way a user can conveniently create various styles of walking by specifying parameters like step length, velocity or stride width. Compared to the other locomotion algorithms above, this approach performs interactively, providing real-time feedback and the ability to produce different styles of locomotion. The method introduced here to animate human running is similar to K LAW in satisfying these criteria. However, a different control scheme is adopted (see section 3) since running is a different motion from walking (see next section) requiring a different set of control parameters (see section 3.1).

2.2 Running Concepts

A bipedal running stride consists of 2 steps as shown in Figure 1. If we assume the motion of the legs to be symmetric we can restrict our analysis to one step, for instance from heel-strike of the left leg to heel strike of the right leg. A running step is made up of a single support state where one foot is off the ground and a flight state where both feet are off the ground. With respect to one stride, each leg cycles through a stance phase and a swing phase shifted in time. Compared with walking where the stance phases for the two legs overlap in running the swing phases overlap.

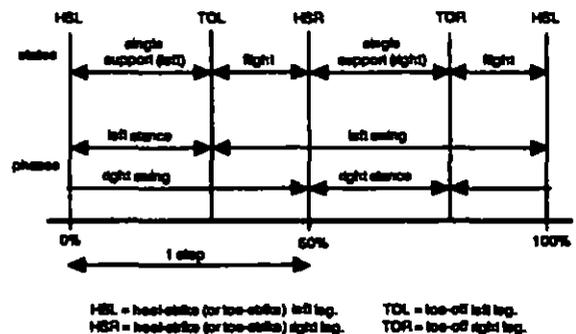


Figure 1 Running stride

A running stride can be characterized by the following four high-level parameters: velocity (v), step length (sl), step frequency (sf) and flight height (H), where

$$v = sl \times sf \quad (1)$$

Humans when told to run at a particular velocity or step length naturally "choose" the other parameters to maintain a comfortable running stride. For a high-level motion control scheme it is therefore crucial that the inter-relationships between these parameters are determined. Research on human running indicates that an increase in velocity is accompanied by a linear increase in step length up to a speed of about 400 m/min (~ 24 km/h), then the step length levels off and only step frequency is increased to further increase velocity [15, 16, 21]. On the other hand, step frequency increases in a curvilinear manner with respect to velocity, small increases in step frequency at lower velocity, and larger increases at higher velocity [10]. This linear relationship between v and sl is now established more formally based on actual human running data from [15, 16, 21, 28]. The formula below was derived by linear regression applied to the data with $v \leq 400$ m/min. A multiple r^2 coefficient of 0.94442 was obtained (for $n = 80$ data pairs) indicating very good correlation (sl is in m and v in m/min)

$$sl = 0.1394 + 0.00465 v$$

To this equation, a term $level$ is now added which models the level of expertise in running based on the observation [10, 21] that better runners have a greater stride length at a given velocity than less skilled or poor runners, where -0.001 (poor) $\leq level \leq 0.001$ (skilled)

$$sl = 0.1394 + (0.00465 + level) v$$

Another factor influencing this relationship between v and sl is leg length. For walking, Inman derived a normalization formula [17] relating step length to body height (which is a function of leg length). For running, we adopt an approach suggested by Alexander [1]. He observed that geometrically similar animals of different sizes have runs which are dynamically similar whenever their speeds made their Froude number equal. The Froude number is defined as the ratio between kinetic and potential energies, $v^2/(2gl)$, where l is the leg length. Since leg length is proportional to body height, the Froude equality can be expressed as $v^2/body_height = v_{1.8}^2/1.8$, where 1.8 m is the default body height. Putting this information into our equation relating velocity to step length, we now have

$$sl = 0.1394 + (0.00465 + level) v \sqrt{\frac{body_height}{1.8}} \quad (2)$$

Figure 2 illustrates equation 2 as well as the real data pairs (crosses). The three lines close together indicate relationships with $level = 0$ and body height equal to 1.7 m (lower), 1.8 m (middle) and 1.9 m (upper). For the two lines with the smallest and largest slope, body height is 1.8 m with $level$ set to minimum for the former and maximum for the latter, respectively.

Equation 2 represents the normalization formula for running. In correspondence with research on running,

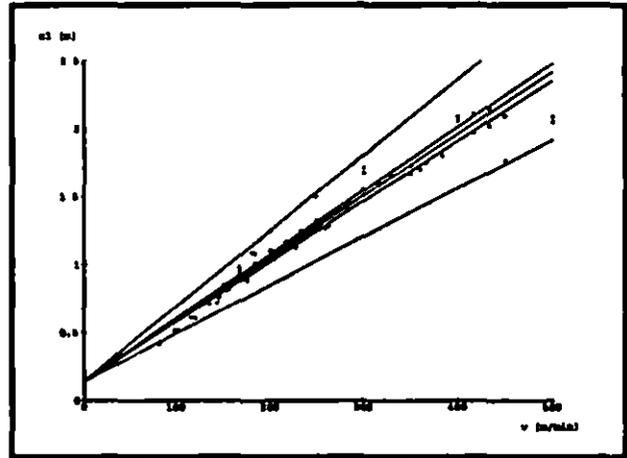


Figure 2 Step length as a function of velocity

we apply this formula up to v_{norm} , where $v_{norm} = 400$ m/min (~ 24 km/h) for a human with a body height of 1.8 m. If v increases further, sl is kept constant (sl_{norm}) and only sf is increased ($v = sl_{norm} \times sf$)

Equation 1 and 2 establish a relationship between v , sl and sf . The fourth running parameter, flight height H , is a function of flight time t_{flight} and the vertical position of the pelvis at toe-off (y_1) and heel-strike (y_2), g denotes the gravitational acceleration

$$H = \left(t_{flight} - \frac{2(y_1 - y_2)}{g t_{flight}} \right)^2 \frac{g}{8} \quad (3)$$

Thus, in order to calculate H the duration of flight needs to be known. Research on human running suggests that an increase in velocity has little effect on the time for flight [15, 21, 24]. However, the time for support in running decreases significantly as the speed is increased [10]. Therefore, an increase in step frequency is due mainly to a decrease in the time for support. Correlations on the running data by [15, 21, 24] suggest that the time of flight initially increases with step frequency, reaches a maximum at about 190 steps/min after which it levels off slightly. A best fit (residual mean square = 0.00087 for $n = 20$ data pairs) was obtained by a 3rd order polynomial. As noted in the literature [21], expert runners tend to have a longer flight period than poor runners at comparable step frequencies. Incorporating a $level$ attribute into the equation similar to above (here, -0.0001 (poor) $\leq level \leq 0.0001$ (skilled)), we have

$$t_{flight} = -8.925 + (0.131 + level) sf - 0.623 \cdot 10^{-3} sf^2 + 0.979 \cdot 10^{-6} sf^3 \quad (4)$$

The running data available ranged about between 160 step/min and 230 steps/min (t_{flight} is in sec). To get a more general expression for the relationship between sf and t_{flight} , extrapolation becomes necessary. From experience, we think that the following quadratic equation provides a good relationship below 180 step/min

$$t_{flight} = -0.675 \cdot 10^{-3} - (0.15 \cdot 10^{-3} + level) \cdot sf + 0.542 \cdot 10^{-5} \cdot sf^2 \quad (5)$$

The above leads to the following calculation of t_{flight} from sf from 0—180 steps/min, equation 5 is used, from 180—230 steps/min, equation 4 is applied above 230 steps/min, t_{flight} is kept constant

Now we are ready to calculate the durations for the leg phases in running. From Figure 1 it follows that

$$\begin{aligned} t_{stance} &= t_{step} - t_{flight}, \\ t_{swing} &= t_{step} + t_{flight}, \end{aligned} \quad (6)$$

where t_{step} is $1/sf$. These durations manifest timing constraints which guarantee natural looking interpolations of the joint angles of a desired running stride. This is explained as part of the running algorithm in the next section

3 Running Algorithm

Our motion control algorithm for human running has been developed with the following four design goals in mind to make it a useful tool for an animator: ease of motion specification, interactivity and real-time feedback generation of believable motion, ease of customizing and personalizing motion. To meet these goals knowledge about human running has been incorporated at various levels. Empirical knowledge is applied to determine the kind of control parameters and how they are interrelated, the four main running parameters introduced in the previous section define a basic running stride while nineteen attributes can be set to individualize a run such as amount of torso tilt, arm swing and choosing between heel strike or toe-strike. Physical knowledge determines how the center of the body moves during a running stride during the support state its motion is defined by an interpolating cubic spline whereas during flight it follows a parabolic trajectory. Limb-coordination knowledge is used to set up both state-constraints for support and flight, and phase-constraints to "guide" the joint-angle interpolation of the stance and swing phases

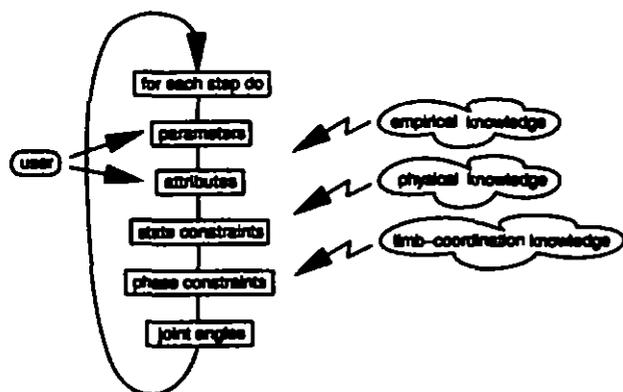


Figure 3 Running algorithm

The basic algorithm is illustrated in Figure 3. The main loop is executed for each running step, which means that changes the user makes to any of the parameters or attributes become active on the next running step. At the end of each step, the current front leg and hind leg are switched to initialize the next step. This way, acceleration and deceleration are possible, as well as starting and stopping which are just special cases of a "normal" step (see section 3.3 below). In the following a closer look is taken at each part of the algorithm in turn.

3.1 Parameters and Attributes

The parameters and attributes of the running algorithm are the interface to the user and control the current running stride. They have default values which can be changed interactively via sliders to customize a run. A distinction between parameters and attributes has been made such that the parameters define the basic running stride while the attributes change the expression or personality of the stride. A change in one of the parameters causes a change in the other parameters to maintain a natural stride as well as in some attributes (see section 3.1.2 below), whereas a change in any of the attributes is independent of the parameters.

3.1.1 Parameters

The parameter panel is illustrated in Figure 4. There are four main parameters: velocity, step length, step frequency and flight height, plus an additional slider for the level of expertise in running as discussed in section 2.2 above. If one of the parameters is changed the other ones are updated using equations 1, 2, 3 and 4 or 5, respectively. The actual calculations of flight height and time of flight are a bit more complex and explained in section 3.2, since we do not yet know the vertical position of the pelvis (kinematic center of the body) at toe-off and impact (y_1 and y_2 in equation 3).

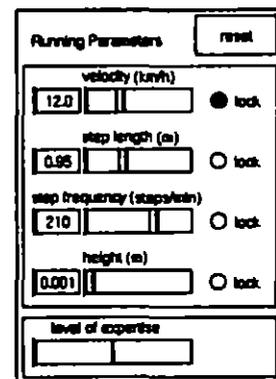


Figure 4 Running parameters

Parameters can be locked to produce somewhat unnatural strides. This is shown in Figure 4, where the velocity was locked at 12 km/h and then step frequency was increased from the default 187 steps/sec to 210, which decreased the normally chosen step length of 1.07 m to 0.95 and the flight height from 0.004 m to 0.001.

3.1.2 Attributes

The attributes allow the user to individualize a running stride. As shown in Figure 5, nineteen attributes have been implemented at this time. Five of these control the movement of the arms and shoulders, two attributes control torso tilt and sway, three attributes are provided to alter the motion of the pelvis and nine attributes change the motion of the legs in one way or other. Among the attributes for the legs are a heel-toe-strike button (heel or toe touches ground first at impact), a bounciness slider (amount of knee bend during support), an overstride slider (amount by which the foot impacts the ground ahead of the body), and a stride-width slider (lateral distance between the feet).

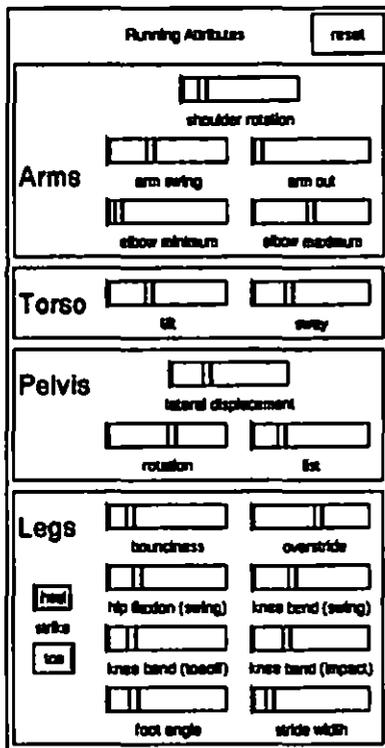


Figure 5 Running attributes

As with the parameters, default values for the attributes are chosen to produce a natural running stride. This involves the values of some attributes being automatically adjusted when a parameter value changes. The adjustment is necessary since there is a large range in the kinematics of running from a slow run to a fast run. For example, the overstride value decreases with increasing velocity, pelvic rotation increases with step length, bounciness increases with flight height, whereas the knee angle at toe-off decreases with step length. These relationships are implemented as linear functions of the current parameters, the attribute values and their extreme values. Some attributes are also coupled. For instance, lateral displacement increases as stride-width increases and both decrease with step frequency. Lastly, there are some "hidden" attributes which can not be set directly by the user but are automatically calculated, for exam-

ple, the ankle and metatarsal angles at toe-off which are functions of step length, or the ankle angle at impact which is a function of the overstride and the heel-toe-strike attributes.

Based on the parameters and attributes settings for the current running step, the state constraints including the motion of the pelvis are now determined, followed by the phase-constraints which guide the interpolation of the joint angles (section 3.3).

3.2 State Constraints

The state constraint principle is illustrated in Figure 6 for a step beginning with heel-strike (subsequently used to mean heel-or-toe-strike) of the right leg and ending at heel strike of the left leg. For the opposite step from heel-strike left leg to heel-strike right leg all the calculations below are mirrored. The constraints establish the leg angles for the stance leg at the beginning of a step (HSR) and at toe-off (TOR), as well as the leg angles of the swing leg at the end of the step (HSL). These internal "keyframes" serve as the basis for interpolating the leg angles in section 3.3.

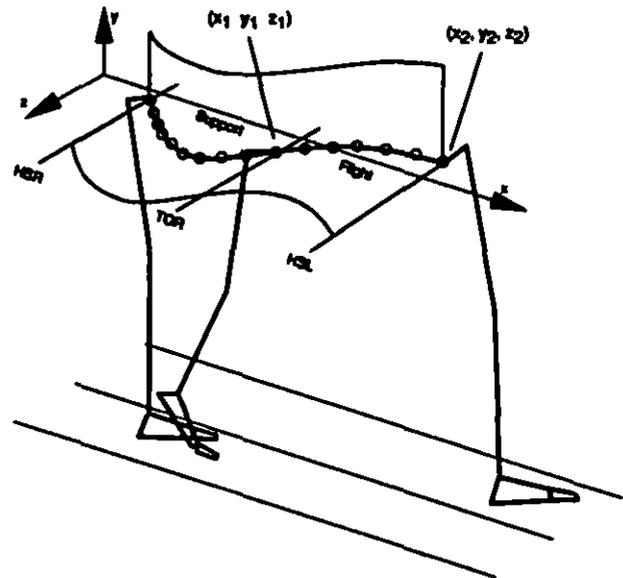


Figure 6 State constraints

We now explain how these constraints are calculated given the current parameters and attributes, assuming that the leg angles at the beginning of the step (HSR) are known from the end of the previous step or the resting position on the initial step. All the computations are done in 3-D. For the explanations below it is noted that according to research on human running, the actual values for lateral displacement of the body at toe-off and heel-strike are 80% (to either side of neutral) of the maximum value given by the attribute. Similarly, pelvic rotation (in the transverse, $x-z$ plane) is a maximum at toe-off and about 20% at heel-strike, whereas pelvic list (in the coronal, $y-z$ plane) is a minimum (zero) at toe-off and about 80% at heel-strike. This is shown in Figure 7. It is also illustrated that at mid-support, lateral displacement as well as pelvic list are a maximum and rotation is a minimum (zero).

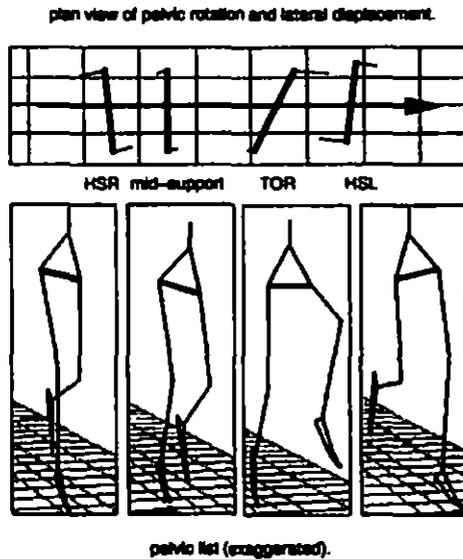


Figure 7 Motion of pelvis for one step

The first stage in calculating the state constraints for the current step is to add the current step length to the heel position at HSR to obtain the heel position at the end of the step (HSL). Then the position of the center of the pelvis at HSL, (x_2, y_2, z_2) , is determined "bottom up" using the current attributes for stride width, overstride, knee bend at heel strike, as well as the percentages of lateral displacement of the body pelvic rotation and list at heel strike. Given the position of the pelvis at heel strike we now calculate "backwards" using information on the flight state to obtain the position and orientation of the toe-off leg (TOR). For this purpose, the length of the toe-off leg (rad) from toe to the center of the pelvis is computed first using the current attribute values. Note that we can not directly calculate the pelvis position at toe-off since the orientation of the leg in the sagittal ($x-y$) plane is unknown (unlike at heel-strike where it is known from the overstride attribute).

Two cases need to be considered now (I) the flight time (t_{flight}) is given from equation 4 and the flight height H is still to be determined (which is the case if any of v , sl , sf were changed by the user), (II) H is given and t_{flight} is still unknown (which is the case if H was changed by the user). In case (I), there is an analytic solution to calculating the pelvis (x_1, y_1, z_1) at toe-off

$$x_1 = x_2 - v t_{flight}, \quad (7)$$

$$y_1 = \sqrt{rad^2 - (x_1 - x_{toe})^2} \quad (8)$$

where x_{toe} is known from the previous step and z_1 is the percentage of lateral displacement at toe-off as explained above. Given y_1 and y_2 , H is obtained from equation 3. For case (II) the pelvis at toe-off is solved numerically by a two-dimensional Newton Raphson method [11] using equations 3 and 8, whereby $\frac{z_2 - z_1}{v}$ is substituted for t_{flight} in the former. Once x_1 is known we determine t_{flight} by equation 7.

With these "keyframes" at HSR, TOR and HSL in place, the translation of the pelvis can now be determined. During flight the pelvis follows a parabolic trajectory, with $0 \leq t \leq t_{flight}$ and z being interpolated between z_1 and z_2

$$x = x_1 + vt$$

$$y = y_1 + \sqrt{2gH}t - \frac{1}{2}gt^2$$

During support, the pelvis moves along an interpolating cubic spline segment whose four control points are at HSR, mid-support, TOR and mid-flight. Whereas the coordinates at HSR, TOR and mid-flight are known from above, the control points for mid-support are chosen as follows from research on human running [8] it is known that the kinetic and potential energy changes within a stride are simultaneous and are both lowest about the middle of support. Assuming that the motion of the whole body is represented by the pelvis, the vertical position of the pelvis at mid-support is a minimum, defined as a function of the vertical pelvis position at HSR and TOR as well as bounciness and flight height, such that the bigger H the lower y , the x -position of the pelvis at mid support is also a minimum (behind average forward position p) where a value of $0.8 \times p$ has given good results. Finally, the z control point of the pelvis at mid-support is set to the maximum lateral displacement as mentioned above. The translation of the pelvis is illustrated in Figure 6, with changes in velocity indicated by differently spaced circles along the path, also shown is lateral displacement of the body.

The calculation of the orientation of the pelvis—rotation and list—during the current step completes the state constraints. This is done by linear interpolation between the four keyframes shown in Figure 7. The positions of the hip for the stance and swing leg are now known and used next to interpolate the leg angles.

3.3 Phase Constraints

Given the constraints on the support and flight states of the current running step introduced in the last section, the phase-constraints further subdivide the constraints with respect to the stance and swing phases of the legs in order to naturally interpolate the joint angles. The subdivisions are based on observations of how real humans run. This is illustrated in Figure 8 and explained below.

3.3.1 Single Support

During single support from HSR to TOR, the stance leg angles (right solid leg in Figure 8) are calculated given the hip and the heel/toe position as well as the leg angles at HSR and TOR. We assume the motion of the foot to be planar (vertical). At the beginning of stance, the foot rotates around the heel (heel-strike) or metatarsal joint (toe-strike) until it is flat on the ground at mid-support. In this phase, the position of the ankle and therefore the ankle-hip distance are known and from this the other leg angles are derived trigonometrically. Subsequently the foot stays flat on the ground until the beginning of

the next sub-phase during stance which lasts until TOR and is triggered by either the body passing through the vertical or the ankle-hip distance becoming bigger than the length of the extended leg. In both cases the heel comes off the ground. This is implemented by interpolating both the metatarsal and knee angles from the current time and values until TOR. Then the circle around the toe with the radius toe-ankle (with current metatarsal angle) is intersected with the sphere around the hip and radius thigh-shank (with current knee angle) to compute the remaining leg angles.

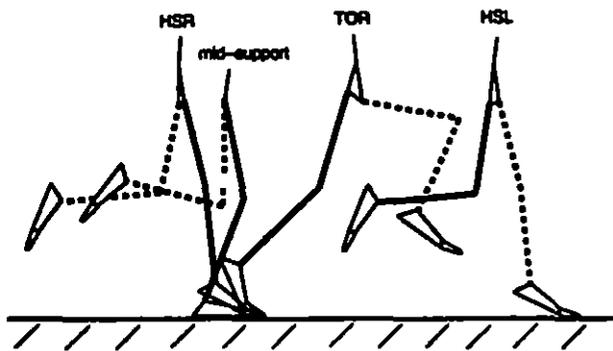


Figure 8 Phase constraints

Automatic recovery procedures during stance are built in, if the hip is too high so the above intersection fails (which can happen, for instance, if bounciness and flight height are significantly reduced), the control point for the pelvis at mid-support is lowered. On the other hand, if the hip is too low (due to an increased bounciness and flight height), the intersection might push the foot through the ground. In this case, the knee angle is automatically temporarily increased.

The swing leg angles during single support (left, dashed leg in Figure 8) are interpolated in the first sub-phase between the values at the end of the previous step (HSR) to mid-support, where the thigh is vertical (sagittal hip angle is zero) and the knee is maximally flexed. The next subphase for the swing leg goes until the end of support (TOR) where the hip is maximally extended and the toe is vertically under the knee. Both the maximum knee flexion and hip extension are functions of the current velocity, but can be adjusted via attribute sliders by the user. If part of the foot stubs the ground during swing (which might occur if bounciness is increased or maximum knee flexion during swing is decreased), an automatic recovery procedure inserts a new control point with temporarily increased knee flexion to lift the foot above the ground. Finally, we note that on the initial step when starting a run from a standing position, the first subphase during swing is omitted and its duration is added to the second phase.

3.3.2 Flight

During flight (TOR to HSL in Figure 8), both legs are in their swing phases. The interpolation of the leg angles for the former stance leg (hind leg) are done such that at HSL, the angles are a percentage (p) of the values at

mid-support of the next step. The percentage automatically varies depending on whether the figure is currently accelerating, running at a constant speed, or decelerating; values for p ranging between 0.6 and 0.9 have given good results. On the last step before stopping, this sub-phase is omitted while the previous stance phases for this leg are extended until HSL. The leg angles for the other leg during flight (dashed leg in Figure 8) are interpolated between their values at TOR and their values heel-strike known from the state-constraints.

3.3.3 Upper Body

Several degrees of freedom of the upper body are animated. Torso tilt and sway in the sagittal plane have default values which can be changed via sliders. Tilt automatically increases with velocity, while sway is interpolated between maximum forward lean reached at mid-support and the maximum backward value at toe-off. In order for the head to always point straight ahead, compensation for pelvis rotation and list are performed in the spine. List compensation is distributed over the five lumbar vertebrae, and rotation over the lumbar and the twelve thoracic vertebrae such that below the seventh thoracic vertebra the rotations are towards the pelvis, whereas above it the rotations are counter to the pelvis. The amount of this counter rotation is adjustable by an attribute.

Arm swing is implemented such that on the forward swing it is equal to the sagittal hip rotation of the opposite (swing) leg multiplied by a default factor adjustable via a slider. On the backward swing, instead of the hip angle which increases and decreases during support, the angle between the vertical and the hip-ankle vector of the opposite leg is used which decreases continuously. The amount of elbow flexion during a running step can also be adjusted by the user through a minimum and maximum flexion attribute. Minimum flexion occurs during the backward swing of the arm at toe-off, maximum flexion during forward swing at toe-off.

This concludes the discussion of the running algorithm. Most of the implementation is based on research and observations on human running, and the main contribution of this approach is that it pulls together different knowledge and techniques to provide an interactive environment in which a user can experiment and animate a wide variety of runs in real-time without having to know about the intricacies of limb-coordination during locomotion.

4 Results

A system called RUNNER has been implemented in C++ according to the principles introduced above. An illustration of the interface is given in Figure 9. The program performs in real-time on a Silicon graphics Indigo² R4000 workstation. For calculations done at 30 frames/sec, this means that an animator can interactively change any of the parameters and attribute sliders while viewing a real-time running stick-figure on the screen. Our model of the human figure has 37 joints and 71 degrees of freedom, and anthropometric data such as body height and relative limb lengths are variable.

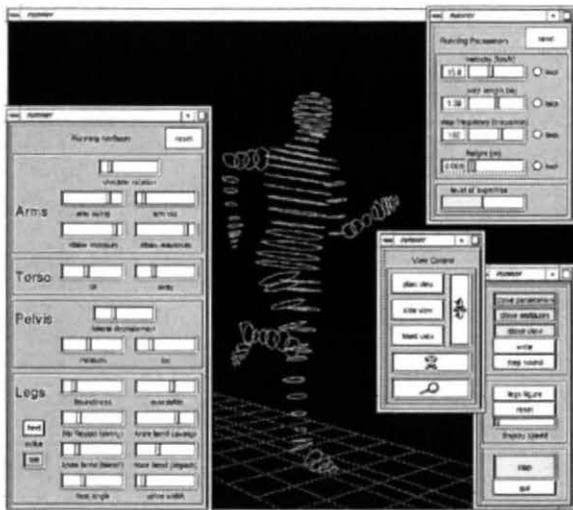


Figure 9: Running interface.

Snapshots of a few sample runs are shown in Figure 10. For example, the top left run was obtained by just reducing the velocity slider to about 5 km/h; the second top run from left was generated by increasing the velocity to about 15 km/h and increasing elbow flexion for both minimum and maximum. The third top run from the left was produced from the settings of the previous run by increasing arm-swing and knee-bend during swing as well as switching from heel-strike to toe-strike. In general, a large number of running styles can be animated. For example, ranges in the parameters from a very slow run at 2 km/h to a fast run at 25 km/h with a step length well over 2 m are possible. Also, moving the attribute sliders to their extreme values results in runs which can look very “stiff” or very “loose”.

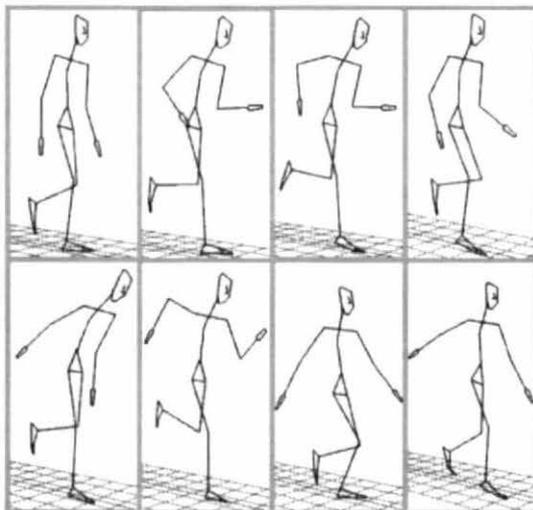


Figure 10: Various sample run snapshots at heel-strike of the right leg.

Figure 11 demonstrates a comparison between a human running on a treadmill and a run generated by our algorithm to match the real run. By setting the body height of our figure to the height of the subject and the velocity to the speed of the treadmill, step length, step frequency and flight height successfully matched with the real run. In addition, the default overstride attribute value was reduced slightly and elbow flexion was increased to closely match the leg and arm movements of the treadmill run.

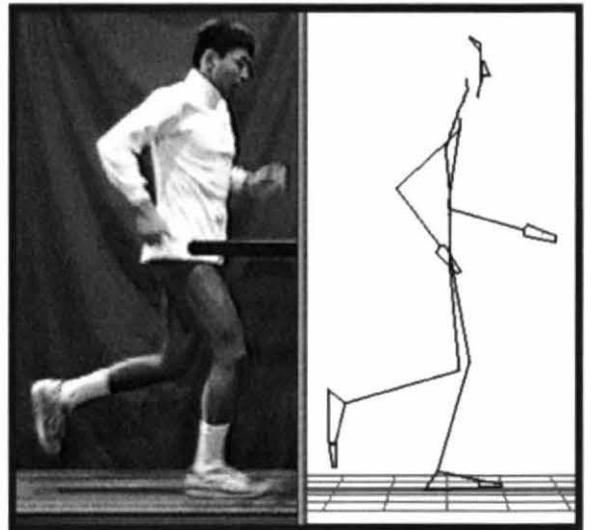


Figure 11: Real treadmill and generated run.

5 Conclusions

A high-level motion control technique has been introduced which allows a user to conveniently create a large variety of human running styles interactively and in real-time. Unlike other high-level techniques, the creative control over the motion remains with the animator. Convincing running animations are achieved by incorporating knowledge on how people run into the algorithm. The approach has proven useful for customizing “real” looking runs or “funny” runs for human-like figures of different sizes and shapes. The results are better than motion-captured data in a sense that true 3-D motion is generated which is easily modifiable on the fly.

We are currently extending the system to running along inclines and arbitrary paths. In this way, the control could be hooked up to devices like joysticks to drive figures in interactive games or virtual environment applications. Also, the keyframes defined by the state and phase constraints can be exported to general-purpose animation systems for rendering or further manipulation.

The initial research introduced here was carried out at Simon Fraser University supported in part by grants from the Natural Sciences and Engineering Research Council of Canada. We are thankful to Dr. Nakatsu and Dr. Mase at ATR MI&C for their continuing support of new developments.

References

- [1] ALEXANDER R M The gaits of bipedal and quadrupedal animals *The International Journal of Robotics Research* 3, 2 (1984), 49-59
- [2] BOULIC R ET AL Human free-walking model for a real time interactive design of gaits In *Computer Animation '90, Proceedings* (April 1990), N Magnenat-Thalmann and D Thalmann, Eds, Springer-Verlag, pp 61-79
- [3] BRUDERLIN A AND CALVERT I Goal-directed, dynamic animation of human walking In *Computer Graphics (SIGGRAPH '89 Proceedings)* (July 1989), vol 23, pp 233-242
- [4] BRUDERLIN A AND CALVERT I Interactive animation of personalized human locomotion In *Graphics Interface '93, Proceedings* (May 1993), pp 17-23
- [5] CALVERT I ET AL Desktop animation of multiple human figures *IEEE Computer Graphics & Applications* 13, 3 (1993), 18-26
- [6] CALVERT I ET AL Towards the autonomous animation of multiple human figures In *Computer Animation '94 Proceedings* (1994), pp 69-75
- [7] CASSELL J ET AL Animated conversation Rule-based generation of facial expression, gesture & spoken intonation for multiple conversational agents In *Computer Graphics (SIGGRAPH 94 Proceedings)* (July 1994), pp 413-420
- [8] CAVAGNA G ET AL Mechanical work in running *Journal of Applied Physiology* 19 (1964), 249-256
- [9] COHEN M Interactive spacetime control for an animation In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), vol 26 pp 293-302
- [10] DILLMAN C Kinematic analysis of running *Exercise and Sport Sciences Reviews* 3 (1975), 193-218
- [11] FLETCHER R *Practical Methods of Optimization* John Wiley & Sons, New York, 1987
- [12] GIRARD M AND MACIEJWESKI A Computational modeling for the computer animation of legged figures In *Computer Graphics (SIGGRAPH '85 Proceedings)* (1985), vol 19, pp 263-270
- [13] HODGINS, J Simulation of human running In *IEEE International Conference on Robotics and Automation, Proceedings* (1994), pp 1320-1325
- [14] HODGINS J ET AL Animating human athletics In *Computer Graphics (SIGGRAPH '95 Proceedings, Los Angeles CA, August 6-11)* (1995), pp 71-78
- [15] HOGBERG P Length of stride, stride frequency, "flight" period and maximum distance between the feet during running with different speeds *Arbeitsphysiologie* 14 (1952), 431-436
- [16] HOSHIKAWA I ET AL Analysis of running pattern in relation to speed *Medicine and Sport Biomechanics III* 8 (1973), 342-348
- [17] INMAN V ET AL *Human Walking* Williams & Wilkins, Baltimore, 1981
- [18] ISAACS P AND COHEN M Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics In *Computer Graphics (SIGGRAPH '87 Proceedings)* (1987), vol 21, pp 215-224
- [19] KO H AND BADLER N Straight line walking animation based on kinematic generalization that preserves the original characteristics In *Graphics Interface '93, Proceedings* (May 1993), pp 9-16
- [20] KOGA Y ET AL Planning motions with intentions In *Computer Graphics (SIGGRAPH '94 Proceedings)* (July 1994) pp 395-408
- [21] KURAKIN M Relationships among running parameters *Yessis Review* 8, 1 (1973), 1-4
- [22] LEE P ET AL Strength guided motion In *Computer Graphics (SIGGRAPH '90 Proceedings)* (1990), vol 24, pp 253-262
- [23] LIU Z ET AL Hierarchical spacetime control In *Computer Graphics (SIGGRAPH 94 Proceedings)* (July 1994), pp 35-42
- [24] NILSSON J, ET AL Changes in leg movements and muscle activity with speed of locomotion and mode of progression in humans *Acta Physiologica Scandinavica* 123, 4 (1985), 457-475
- [25] PHILLIPS C AND BADLER N Interactive behaviors for bipedal articulated figures In *Computer Graphics (SIGGRAPH '91 Proceedings)* (1991), vol 25, pp 359-362
- [26] RAIBERT M AND HODGINS, J Animation of dynamic legged locomotion In *Computer Graphics (SIGGRAPH '91 Proceedings)* (1991), vol 25, pp 349-358
- [27] STEWART J AND CREMER, J Beyond keyframing An algorithmic approach to animation In *Graphics Interface '92, Proceedings* (May 1992) pp 273-281
- [28] WEBER W AND WEBER E *Mechanics of the Human Walking Apparatus* Springer-Verlag, Berlin, 1992
- [29] WILHELMS J Using dynamic analysis to animate articulated bodies such as humans and robots In *Graphics Interface '85, Proceedings* (1985), pp 97-104
- [30] WILHELMS J Virya-a motion control editor for kinematic and dynamic animation In *Graphics Interface '86, Proceedings* (1986), pp 141-146

Motion Signal Processing

Armin Bruderlin¹
Simon Fraser University

Lance Williams²
Apple Computer Inc

Abstract

Techniques from the image and signal processing domain can be successfully applied to designing, modifying, and adapting animated motion. For this purpose, we introduce multiresolution motion filtering, multitarget motion interpolation with dynamic time warping, waveshaping, and motion displacement mapping. The techniques are well suited for reuse and adaptation of existing motion data such as joint angles, joint coordinates, or higher level motion parameters of articulated figures with many degrees of freedom. Existing motions can be modified and combined interactively and at a higher level of abstraction than conventional systems support. This general approach is thus complementary to keyframing, motion capture, and procedural animation.

Keywords: human animation, motion control, digital signal processing

1 Introduction

Motion control of articulated figures such as humans has been a challenging task in computer animation. Using traditional keyframing [27], it is relatively straightforward to define and modify the motion of rigid objects through translational and rotational trajectory curves. However, manipulating and coordinating the limbs of an articulated figure via keyframes or the spline curves they define is a complex task that draws on highly developed human skills. More general, global control of the character of an animated motion would be useful in fine tuning keyframed sequences. Such global control would make predefined sequences more useful, and libraries of animated motion more valuable.

Much of the recent research in motion control of articulated figures has been directed towards reducing the amount of motion specification to simplify the task of the animator. The idea is to build some knowledge about motion and the articulated structure into the system so that it can execute certain aspects of movement autonomously. This has led to the development of higher level control schemes [5, 6, 15, 22, 33] where the knowledge is frequently specified in terms of rules, and physically based modeling techniques [8, 12, 18, 30, 31] in which knowledge is embedded in the equations of motion, constraints, and possibly an optimization expression. Both approaches often suffer from lack of interactiv-

ity: they don't always produce the motion which the animator had in mind, and complex models have a slow interactive cycle. To increase the expressive power of such models, more control parameters can be introduced. Once again, higher level editing tools for the trajectories of such control parameters would ease animators' burdens and generalize their results.

An alternative method to obtain movements of articulated figures is performance animation where the motion is captured from live subjects. Although a variety of technologies have been developed to fairly reliably measure performance data [19], the computer graphics literature makes scant mention of editing techniques for recorded motion. In the absence of effective editing tools, a recorded movement that is not quite "right" requires the whole data capture process to be repeated.

Because of the complexity of articulated movements and the limitations of current motion control systems as outlined above, we believe that it is desirable to develop tools that make it easy to reuse and adapt existing motion data. For this purpose, we adopt techniques from the image and signal processing domain which provide new and useful ways to edit, modify, blend, and align motion parameters of articulated figures. These techniques represent a pragmatic approach to signal processing by providing analytic solutions at interactive speeds, and lend themselves to higher level control by acting on several or all degrees of freedom of an articulated figure at the same time.

In this paper, we treat a motion parameter as a sampled signal. A signal contains the values at each frame¹ for a particular degree of freedom. These values could come from evaluating a spline curve in a keyframing system, or be derived from the tracked markers in a motion capture system. In animating articulated figures, we are often concerned with signals defining joint angles or positions of joints, but the signal processing techniques we have implemented also apply to higher level parameters like the trajectory of an end effector or the varying speed of a walking sequence.

In Section 2, we present the method of multiresolution filtering and its application to parameters of motion. Section 3 discusses multitarget interpolation, while pinpointing a severe problem of this technique when used for motion blending — the absence of an automatic alignment or registration of movements. A solution to this problem is given based on the principle of dynamic time warping. Section 4 introduces waveshaping as a rapid nonlinear signal modification method useful for tasks such as mapping joint limits of articulated figures. Section 5 concludes the editing techniques we have developed with motion displacement mapping, an extremely general tool which permits editing of densely sampled motion data with the ease of keyframing. Each of these sections provides illustrative examples. Finally, conclusions are given in section 6.

¹ Sampled signals have values defined at regular intervals which, in a good animation system, should be completely decoupled from the nominal frame rate of the final product. We will speak of "frames" at the sample rate without intending any loss of generality.

¹School of Computing Science, Simon Fraser University, Burnaby, B.C. V5A 1S6 Canada, (armin@cs.sfu.ca)

²Apple Computer Inc, 1 Infinite Loop, MS 301, 3J, Cupertino, CA 95014, USA (lance@nca.aol.com)

2 Multiresolution Filtering

In the motion capture realm, most systems have provision for non-linear impulse-noise removal filters (Tukey filters) as well as linear smoothing filters for noise reduction in digitized data. There has been less published discussion of the use of signal processing operations to edit or modify captured motion for creative purposes. The "lag, drag, and wiggle" recursive filters in Inkwel [17] represent more relevant previous work in the application of signal processing to keyframed 2D animated motion. These filters were used to stylize motion by invoking linear systems behavior without a more structured physical model, and permitted lively animated effects without unduly taxing the animator. In another related approach, Unuma et al. [28] apply Fourier transformations to data on human walking for animation purposes. Based on frequency analysis of the joint angles, a basic 'walking' factor and a 'qualitative' factor like "brisk" or "fast" are extracted. These factors are then used to generate new movements by interpolation and extrapolation in the frequency domain, such that now a walk can be changed continuously from normal to brisk walking.

"Multiresolution filtering" describes a range of digital filter-bank techniques which typically pass a signal through a cascade of lowpass filters to produce a set of short-time bandpass or low-pass signal components. By applying filtering recursively to the output of successive filter bank stages, and downsampling lowpass components as appropriate, these filter banks can be quite efficient; they can produce short-time spectra at roughly the same $n \log(n)$ expense as the Fast Fourier Transform.

The method of multiresolution filtering has been extensively exercised by Burt et al. [4, 20] as an image representation method advantageous for certain kinds of operations, such as seamless merging of image mosaics and intra-image interpolation (noise removal). It has also been applied to temporal dissolves between images [26]. Images may be stored as lowpass (Gaussian) or bandpass (Laplacian) pyramids of spatial filterbands, where each level represents a different octave band of spatial frequencies. Operations like merging two images are then performed band-by-band before reconstructing the image by adding up the resulting bands. In this way, the fine detail of an image corresponding to the higher frequencies can be treated separately from the coarse image features encoded by the low frequencies.

In the currently popular wavelet parlance [7], Burt's Gaussian pyramid is a multiresolution analysis in terms of a cubic B-spline scaling function. The corresponding Laplacian pyramid is simply a bandpass counterpart, where each successively higher level of detail has an interpolated copy of the level beneath subtracted from it. The Laplacian pyramid can be computed directly in this way, or via a modified wavelet transform. Burt's method is more efficient for signals of more than one dimension [29]. As a general observation, for synthesis and modification (as well as many analysis tasks for computer vision), oversampled filter banks like Burt's are more useful than strict subband decompositions (where the number of coefficients does not exceed the number of samples in the original signal). A direct contrast is in the way small translations of an image are projected: sparse decompositions change radically with small offsets of the input image, whereas the Burt pyramids change smoothly. The reduction in coefficients attendant on a sub-band filterbank may speed numerical solution of some problems; a recent effort in the animation domain is the wavelet formulation of spacetime interpolation for physically-based keyframing by Liu et al. [18]. They did not use the frequency decomposition to provide direct manipulation of motion, and we believe Burt's method is more appropriate for this purpose.

The first step in applying Burt's multiresolution analysis is to obtain the lowpass pyramid by successively convolving the image with a B-spline filter kernel (e.g. 5×5), while the image is subsampled by a factor of 2 at each iteration (as shown at the left of

Figure 1, where G_0 is the original image). This process is repeated until the image size is reduced to one pixel, which is the average intensity, or DC value. The bandpass pyramid is then calculated by repeatedly differencing 2 successive lowpass images, with the subtrahend image being expanded first in each case (right of Figure 1, where L_0 is the highest frequency band). The image can be reconstructed without manipulation by adding up all the bandpass bands plus the DC. The same procedure can be performed on two or more images at the same time, whereby operations like merging are executed band by band before reconstructing the final result.

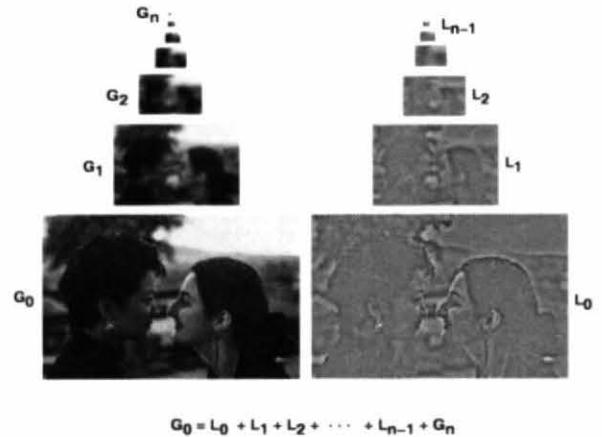


Figure 1: Left: lowpass pyramid; right: bandpass pyramid.

2.1 Motion Multiresolution Filtering

The principles of image multiresolution filtering are now applied to motion parameters of an articulated figure, motivated by the following intuition: low frequencies contain general, gross motion patterns, whereas high frequencies contain detail, subtleties, and (in the case of digitized motion) most of the noise. Each motion parameter is treated as a one-dimensional signal from which the lowpass (G) and bandpass (L) levels are calculated. An example is illustrated in Figure 2 based on the signal of the sagittal knee angle of two walking cycles generated with GAITOR [2].

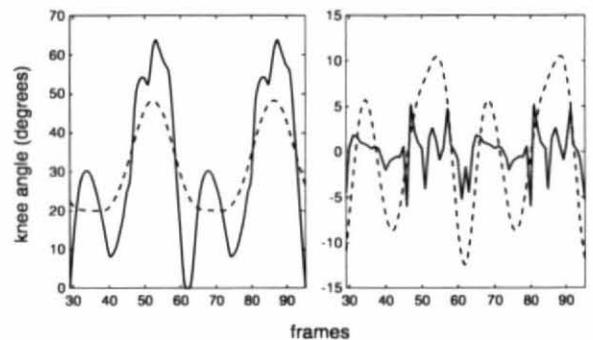


Figure 2: Left: lowpass G_0 (solid) and G_3 (dashed; B-spline kernel of width 5); right: bandpass L_0 (solid) and L_2 (dashed) of the sagittal knee angle for two walking cycles.

2.1.1 Filtering Algorithm

The length m (number of frames) of each signal determines how many frequency bands (fb) are being computed:

$$\text{let } 2^n \leq m \leq 2^{n+1}, \text{ then } fb = n.$$

Instead of constructing a pyramid of lowpass and bandpass sequences where each successive sequence is reduced by a factor of two, alternatively the sequences are kept the same length and the filter kernel (w) is expanded at each level by inserting zeros between the values of the filter kernel (a, b, c below) [3]. For example, with a kernel of width 5,

$$\begin{aligned} w_1 &= [c \ b \ a \ b \ c], \\ w_2 &= [c \ 0 \ b \ 0 \ a \ 0 \ b \ 0 \ c], \\ w_3 &= [c \ 0 \ 0 \ b \ 0 \ 0 \ a \ 0 \ 0 \ b \ 0 \ 0 \ c], \text{ etc.}, \end{aligned}$$

where $a = 3/8, b = 1/4$ and $c = 1/16$. Since we are dealing with signals rather than images, the storage penalty compared to a true pyramid is not as significant ($fb \times i$ versus $4/3 \times i$, where i = number of data points in original signal), while reconstruction is faster since the signal does not have to be expanded at each level. We now state the motion multiresolution algorithm in detail. Steps 1 to 5 are performed simultaneously for each motion parameter signal:

1. calculate lowpass sequence of all fb signals ($0 \leq k < fb$) by successively convolving the signal with the expanded kernels, where G_0 is the original motion signal and G_{fb} is the DC:

$$G_{k+1} = w_{k+1} \times G_k;$$

This can be calculated efficiently by keeping the kernel constant and skipping signal data points (i ranges over all data points of a signal):²

$$G_{k+1}(i) = \sum_{m=-2}^2 w_1(m) G_k(i + 2^k m);$$

2. obtain the bandpass filter bands ($0 \leq k < fb$):

$$L_k = G_k - G_{k+1};$$

3. adjust gains for each band and multiply L_k 's by their current gain values (see example below).
4. blend bands of different motions (optional, see multitarget interpolation below).
5. reconstruct motion signal:

$$G_0 = G_{fb} + \sum_{k=0}^{fb-1} L_k.$$

²We implemented several treatments of the boundary of the signal, that is when $i + 2^k m$ lies outside the domain of the signal. The two most promising approaches have proved to be reflecting the signal, and keeping the signal values constant (i.e. equal to the first/last data point) outside its boundaries.

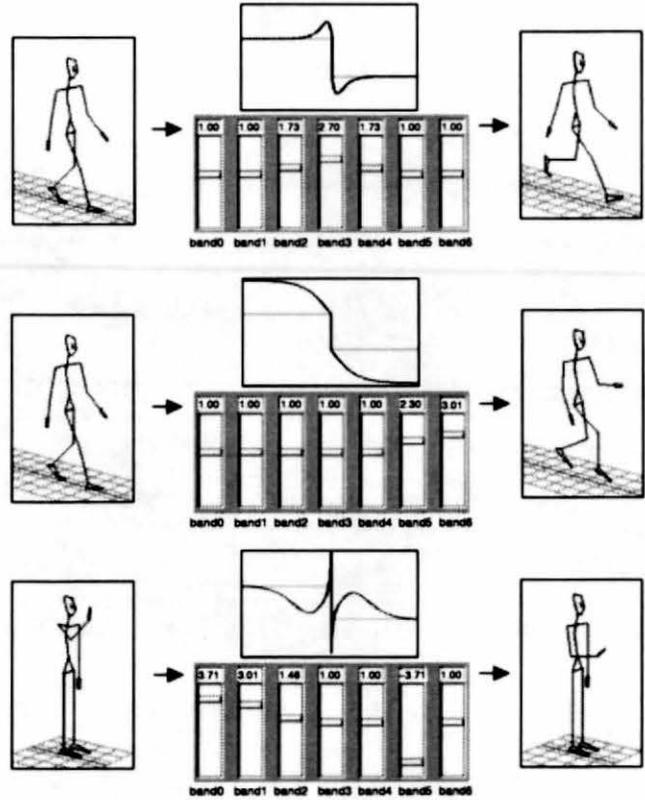


Figure 3: Adjusting gains of bands for joint angles; top: increasing middle frequencies; middle: increasing low frequencies; bottom: using negative gain value.

2.1.2 Examples

An application of motion multiresolution filtering is illustrated in Figure 3. Displayed like an equalizer in an audio amplifier, this is a kind of graphic equalizer for motion, where the amplitude (gain) of each frequency band can be individually adjusted via a slider before summing all the bands together again to obtain the final motion. A step function shows the range and effect of changing frequency gains. We applied this approach successfully to the joint angles (70 degrees of freedom) of a human figure. The same frequency band gains were used for all degrees of freedom. In the example illustrated at the top of Figure 3, increasing the middle frequencies (bands 2, 3, 4) of a walking sequence resulted in a smoothed but exaggerated walk. By contrast, increasing the high frequency band (band 0) added a nervous twitch to the movement (not shown in Figure 3), whereas increasing the low frequencies (bands 5, 6) generated an attenuated, constrained walk with reduced joint movement (Figure 3 middle). Note that the gains do not have to lie in the interval $[0, 1]$. This is shown at the bottom of Figure 3, where band 5 is negative for a motion-captured sequence of a figure knocking at the door, resulting in exaggerated anticipation and follow-through for the knock. We also applied the same filtering to the joint positions (147 degrees of freedom) of a human figure. Increasing the gains for the middle frequency bands of a walking sequence produced a slight scaling effect of the end effectors, and resulted in a squash-and-stretch cartoon walk (Figure 4).

From the examples, it becomes apparent that some constraints such as joint limits or non-intersection with the floor can be violated in the filtering process. Our motion-editing philosophy is to employ constraints or optimization after the general character of the motion has been defined (see displacement mapping in section 5 below; or a

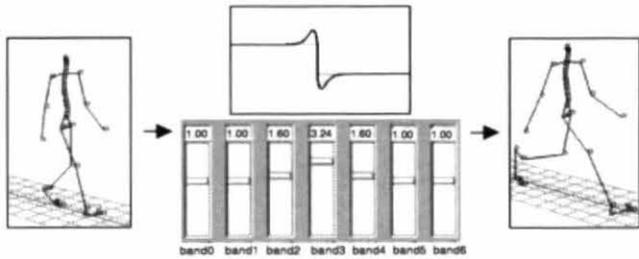


Figure 4: Adjusting gains of bands for joint positions.

more general optimization method [13]). Whereas being trapped in local minima is the bane of global optimization for most problems, animated motion is a good example of an underconstrained problem where the closest solution to the animator's original specification is likely the best. Of course, many animators disdain consistent physics, which is another good reason to decouple motion editing from constraint satisfaction.

Finally, we suggest that a multiresolution approach could also be quite useful in defining motion sequences, rather than simply modifying them. Much like an artist creating a picture blocks out the background first with a big brush, then adds more and more detail with finer and finer brushes, a generic motion pattern could be defined first by low frequencies, and then "finetuned" by adding in higher frequency refinements³.

3 Multitarget Interpolation

Multitarget interpolation refers to a process widely used in computer animation to blend between different models. The technique was originally applied in facial animation [1, 21]. We might have a detailed model of a happy face, which corresponds parametrically to similar models of a sad face, quizzical face, angry face, etc. The control parameters to the model might be high level (like "raise left eyebrow by 0.7"), very high level (like "be happy"), or they might simply be the coordinates of the points on a surface mesh defining the shape of part of the face. By blending the corresponding parameters of the different models to varying degrees, we can control the expression of the face.

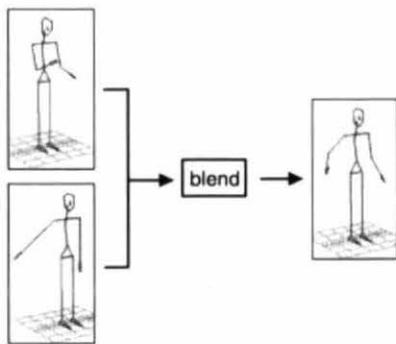


Figure 5: Example of multitarget motion interpolation.

3.1 Multitarget Motion Interpolation

We can apply the same technique to motion. Now we might have a happy walk, a sad walk, angry walk, etc., that can be blended freely to provide a new result. Figure 5 shows an example of blending two

different motions of a human figure, a drumming sequence and a "swaying arm sideways" sequence. In this case, the blend is linear, i.e. add 0.4 of the drum and 0.6 of the arm-sway. In general, the blend can be animated by "following" any trajectory in time. Guo et al. [11] give a good discussion of this approach which they term parametric frame space interpolation. Our approach generalizes on theirs in that the motion parameters such as joint angles to be blended are completely decoupled from one another, and have no implicit range limits. Each component of an arbitrary ensemble of input parameters can have an independent blending coefficient assigned to it.

As indicated in step (4) of the multiresolution algorithm above, we can mix multitarget interpolation and multiresolution filtering to blend the frequency bands of two or more movements separately. This is illustrated in Figure 6 for the same two motions (a drum and an arm-sway) as in Figure 5. Adjusting the gains of each band for each motion and then blending the bands provides finer control while generating visually much more pleasing and convincing motion.

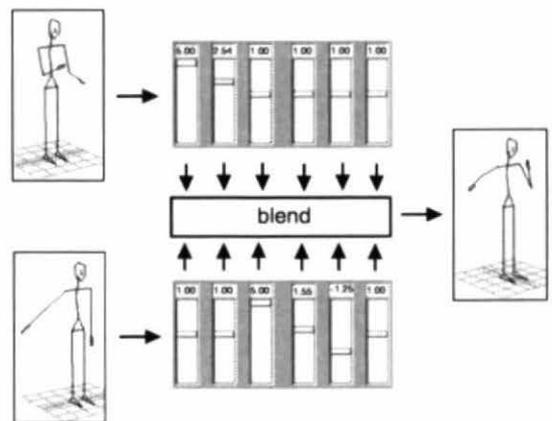


Figure 6: Multitarget interpolation between frequency bands.

However, there is a potential problem when applying multitarget interpolation to motion which relates to the notion of parametric correspondence as stated above: for all our face models to "correspond parametrically" implies that the parameters of each of the models has a similar effect, so that if a parameter raises the left eyebrow of face number one, a corresponding parameter raises the left eyebrow in face number two. If our parameters are simply surface coordinates, it means that the points on each surface correspond, so if the point at U, V coordinates $U1, V1$ is at the tip of the left eyebrow, the point at the same coordinates in any other face will also be at the tip of the left eyebrow.

In motion, parametric correspondence means much the same thing, except that now a correspondence with respect to time is required. If we are blending walk cycles, the steps must coincide so that the feet strike the ground at the same time for corresponding parameter values. If the sad walk is at a slower pace than the happy walk, and we simply blend them together without first establishing a correspondence between the steps, the blend will be a curious dance of uncoordinated motions, and the feet will no longer strike the ground at regular intervals; indeed, they are no longer guaranteed to strike the ground at all (see Figure 7). Thus, multitarget motion interpolation must include both a distortion (remapping a function in time) and a blend (interpolating among different mapped values). In the visual domain a transformation like this is termed a "morph."

Another example is illustrated in Figure 8; here the motion sequences of two human figures waving at different rates and intensities (a "neutral" and a "pronounced" wave) were first blended without timewarping. This resulted in a new wave with undesirable

³Personal communication, Ken Perlin, New York University, 1994.

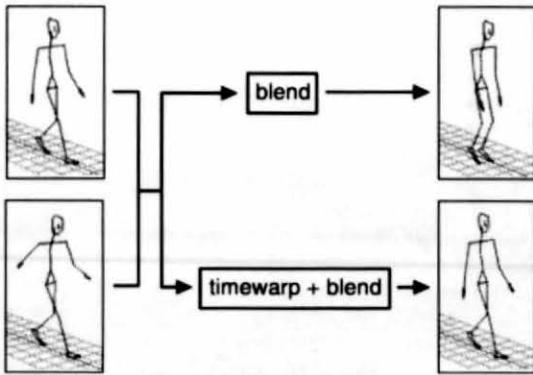


Figure 7: Blending two walks without (top) and with (bottom) correspondence in time.

secondary waving movements superimposed. After timewarping the neutral to the pronounced wave, the blend produced the neutral wave at the pronounced rate. In the following section we describe an automatic method for establishing correspondence between signals to make multitarget motion interpolation meaningful and useful.

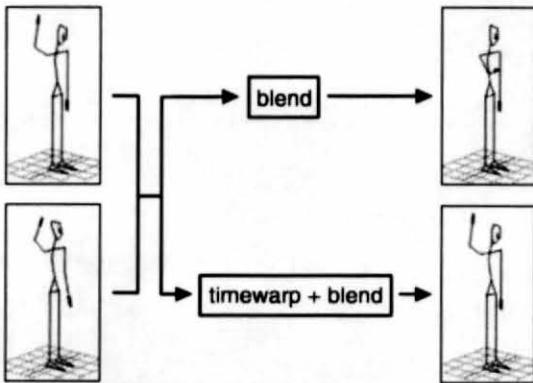


Figure 8: Blending two waves without (top) and with (bottom) correspondence in time.

3.2 Dynamic Timewarping

The field of speech recognition has long relied on a nonlinear signal matching procedure called "dynamic timewarping" to compare templates (for phonemes, syllables or words) with input utterances [9]. Apart from being subject to the usual random error, each acoustic input signal also shows variations in speed from one portion to another with respect to the template signal. The timewarp procedure identifies a combination of expansion and compression which can best "warp" the two signals together.

In our case, timewarping is applied in the discrete time domain to register the corresponding motion parameter signals such as joint angles. In Figures 7 and 8, the timewarping was done simultaneously for all 70 rotational degrees of freedom of the human figure for the duration of the movement sequences. If we have a military march and a drunken stagger, two new gaits can immediately be defined from the timewarp alone: the military march at the drunken pace, and the drunken stagger at the military pace. Figure 9 shows an example for one degree of freedom (knee angle) for the two walks warped in Figure 7. However, we are not limited to these two extreme warps, but may freely interpolate between the mappings of the two walks, and between the amplitudes of the signals through these mappings independently.

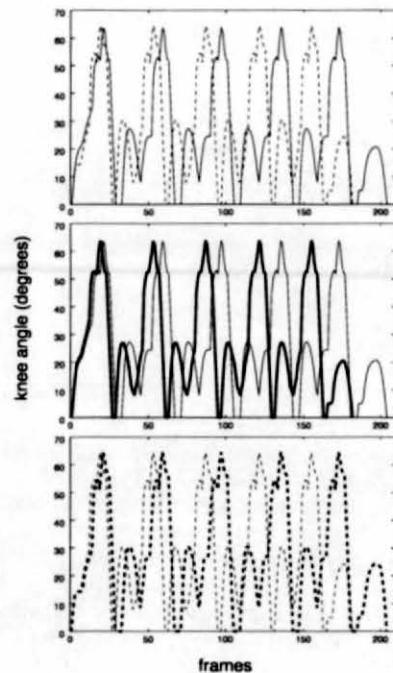


Figure 9: Top: sagittal knee angles curves of two walks; middle: bold = solid curve warped to match dashed; bottom: bold dashed = dashed curve warped to match solid.

3.2.1 Timewarp Algorithm

The problem can be decomposed and solved in two steps: finding the optimal sample correspondences between the two signals, and applying the warp. The vertex correspondence problem is defined as finding the globally optimal correspondence between the vertices (samples) of the two signals: to each vertex of one signal, assign (at least) a vertex in the other signal such that a global cost function measuring the "difference" of the two signals is minimized. In this sense, the problem is related to contour triangulation [10] and shape blending [24], and is solved by dynamic programming optimization techniques. The solution space can be represented as a two-dimensional grid, where each node corresponds to one possible vertex assignment (see Figure 10). The optimal vertex correspondence solution is illustrated in the grid by a path from $(0, 0)$ to $(9, 9)$. In general, there are $O(n^n / n!)$ such possible paths⁴.

When applying timewarping to recognition, the best fit to a canon of signals is computed; no subsequent use is made of a warped signal. The algorithms which perform the warp typically do so by forming a discrete, point-sampled correspondence [9]. For synthetic purposes, more continuous transformations and cost functions are appropriate [32, 25]. We adopted Sederberg's shape blending algorithm [24] which guarantees a globally optimal solution by visiting every node in the grid once ($O(n^2)$ with constant amount of work per node). Upon reaching node (n, n) , the optimal solution is recovered by backtracking through the graph. Sederberg's "physically-based" approach measures the difference in "shape" of the two signals by calculating how much work it takes to deform one signal into the other. The cost function consists of the sum of local stretching and bending work terms, the former involving two, the latter three adjacent vertices of each signal. Intuitively, the larger the difference in distance between two adjacent vertices of

⁴This holds for the vertex correspondence problem, where we favor a diagonal move in the graph over a south-followed-by-east-move or an east-followed-by-a-south-move. For contour triangulation [10], where diagonal moves are denied, the complexity is $O((2n)! / (n!n!))$.

cost function terms:

- stretching work between 2 adjacent vertices in signal (difference in segment lengths)
- bending work between 3 adjacent vertices in signal (difference in angles).

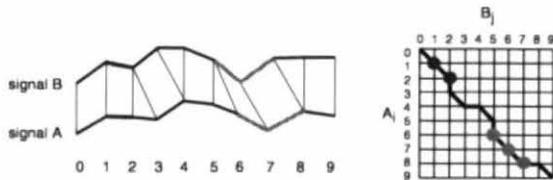


Figure 10: Vertex correspondence problem and cost functions.

one signal and the two vertices of the other (given by two adjacent nodes in the graph), the bigger the cost. Similarly, the larger the difference in angles between three adjacent vertices of one signal and the three vertices of the other (given by three adjacent nodes in the graph), the bigger the cost (for details, see [24]; an illustration is given in Figure 10). One additional check was introduced to make sure that we are really comparing corresponding angles in the two signals: if the middle of the three vertices used to calculate the angle is a local minimum in one signal and a local maximum in the other signal, then one of the angles (α) is inverted before calculating the cost term ($\alpha = 360 \text{ deg} - \alpha$).

The second part of the problem is to apply the warp given the optimal vertex correspondences. As in speech recognition [9], three cases are distinguished: substitution, deletion and insertion. This is indicated in the optimal path by a diagonal, horizontal and vertical line, respectively, between two nodes. For the following explanations, we assume that signal B is warped into A as shown in Figure 11, and the warped signal is denoted by B_w . Then if B_j and A_i are related by a substitution it follows that $B_{w_j} = B_j$. In case of a deletion, where multiple samples of B , ($B_j, B_{j+1}, \dots, B_{j+k}$), correspond to one A_i , $B_{w_j} = \text{mean}(B_j, B_{j+1}, \dots, B_{j+k})$. Finally, an insertion implies that one sample of B , B_j , maps to multiple samples of A , ($A_i, A_{i+1}, \dots, A_{i+k}$). In this case, the values for $B_{w_i}, B_{w_{i+1}}, \dots, B_{w_{i+k}}$ are determined by calculating a cubic spline distribution around the original value B_j .

substitution: 1:1 correspondence of successive samples
 deletion: multiple samples of B map to a sample of A.
 insertion: a sample of B maps to multiple samples of A.

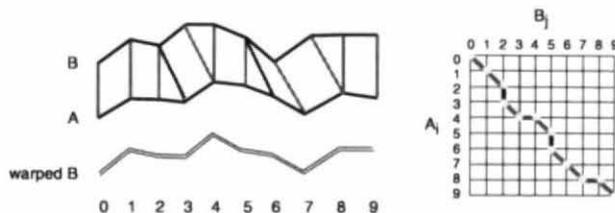


Figure 11: Application of timewarp (warp B into A).

4 Waveshaping

The transformations discussed so far are operations on the time history of a signal. Operations which are evaluated at each point in the signal without reference to its past or future trajectory are occasionally termed *point processes*. Such operations include scaling or offsetting the signal, but are more generally described as a functional composition. Familiar uses of functional composition in graphics include gamma correction and color-lookup, as well as tabular warping functions for images.

“Digital waveshaping” is the term applied to functional composition in computer sound synthesis. In this domain, a normalized input signal x (e.g. scaled to the range from -1 to $+1$) is directed through a discrete *shaping* function f (or waveshaping table) to synthesize steady-state or time-varying harmonic sound spectra. Although waveshaping is in general a nonlinear operation, its effects when applied to an input sine wave can be easily characterized [16]. In practical terms, if f is defined as the identity function $f(x) = x$, the signal will pass through unchanged. If f is slightly changed, say, to having a subtle bump near 0, then the signal x will be altered in that it will have slightly positive values where, and around where, it was zero before, thus x has now some bumps as well. If f is defined as a partial cycle of a cosine function going from minimum to maximum over the $[-1, +1]$ range, the values of x will be exaggerated in the middle and attenuated at the extremes. If f is a step function, x will be quantized to two values.

4.1 Motion Waveshaping

An example of how this idea can be adopted for animation is illustrated in Figure 12. Here the default identity shaping function has been modified to limit the joint angles for a motion sequence of an articulated figure waving. In the figure, “hard” limits are imposed: values of x greater than a limit value simply map to that value. An alternative is a “soft” limit: as values exceed the limit, they are mapped to values that gradually approach it. The implementation of our shaping function is based on interpolating cubic splines [14]; a user can add, delete and drag control points to define the function and then apply it to all or some degrees of freedom of an articulated figure.

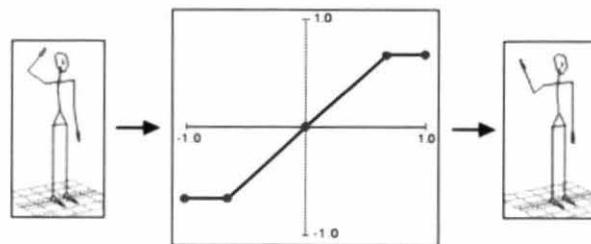


Figure 12: Capping of joint angles via a shape function.

Another application of waveshaping is to map the shape of input motions to a “characteristic” function. The shaping function in Figure 13 applied to the motion-captured data of a human figure sitting and drinking introduced extra undulations to the original monotonic reaching motion. In this way, it is possible to build up a library of shaping functions which will permit rapid experimentation with different styles of movement.

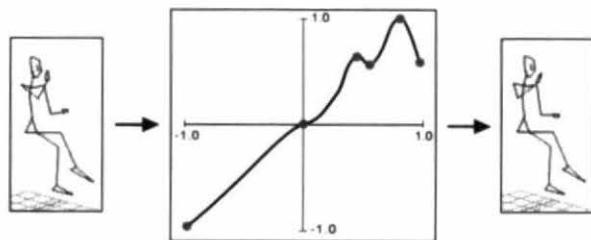


Figure 13: Adding undulations to motion via waveshaping.

5 Motion Displacement Mapping

Displacement mapping provides a means to change the shape of a signal locally through a displacement map while maintaining continuity and preserving the global shape of the signal. To alter a movement, the animator just changes the pose of an articulated figure at a few keyframes. A spline curve is then fitted through these displacements for each degree of freedom involved, and added to the original movement to obtain new, smoothly modified motion. The basic approach is illustrated in Figure 14. Step 1 is to define the desired displacements (indicated by the three vertical arrows) with respect to the motion signal; in step 2, the system then fits an interpolating cubic spline [14] through the values of the displacements (note that the first and last data points are always displacement points). The user can then adjust the spline parameters in step 3 before the system calculates the displaced motion satisfying the displacement points (step 4).

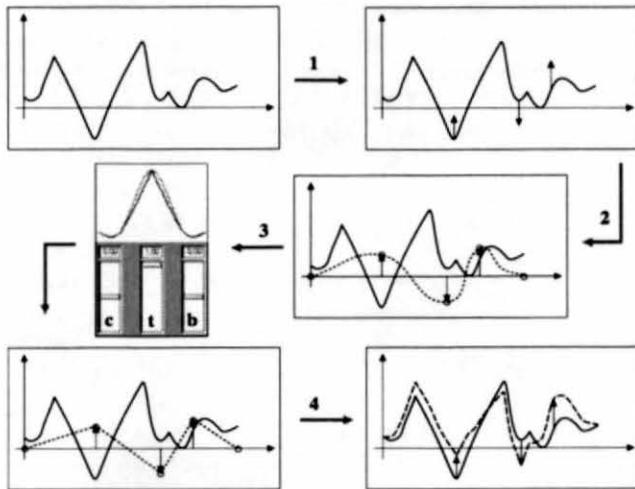


Figure 14: Steps in displacement mapping.

The displacement process can be applied iteratively until a desired result is achieved. Since the operation is cheap, a fast feedback loop is guaranteed. In the top part of Figure 15, we took the output of a multiresolution filtering operation on joint angles of a human walking figure, where some of the joint limits were violated and the feet did not make consistent contact with the ground, and read it into LifeForms [5], a system to animate articulated figures. There we adjusted some of the joints and translated the figure at a few keyframes for which displacement curves were quickly generated and applied to the motion of the figure as described above. To refine the resulting motion, a second loop was executed; a frame of the final result is shown on the top right of Figure 15. The same technique was used in modifying the rotoscoped motion of a human figure sitting and drinking (Figure 15, middle). Here, three out of the 600 motion-captured frames were modified to include some additional gestures of the arms and legs. In Figure 15, bottom, the joint angles for the arm and neck of a motion-captured knocking-at-a-door-sequence were changed for one frame via motion displacement mapping to obtain a knock at a higher impact point.

6 Conclusions

In this paper we have assembled a simple library of signal processing techniques applicable to animated motion. A prototype system has been implemented in the programming language C using the

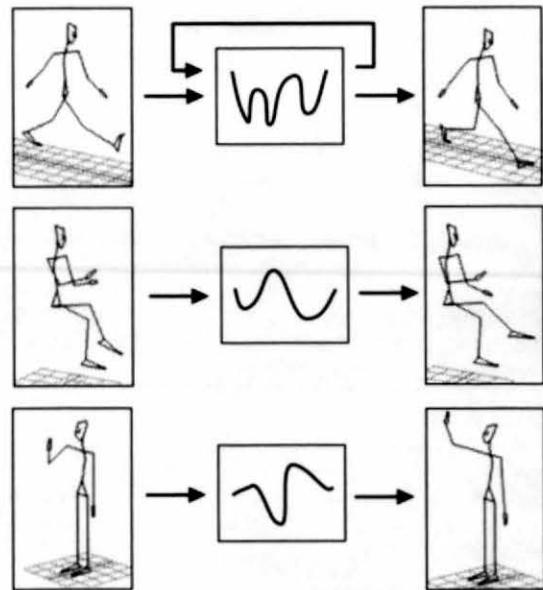


Figure 15: Examples of applying displacement curves.

Khoros application development environment [23]. The immediate goals of our motion-editing experiments have been fulfilled: the motion signal processing techniques provide a rapid interactive loop, and facilitate reuse and adaptation of motion data. By automating some aspects of motion editing such as time-registration of signals or increasing the middle frequencies for several degrees of freedom at the same time, these techniques lend themselves to higher level motion control and can serve as building blocks for high-level motion processing.

Of all the techniques introduced here, perhaps motion displacement mapping will prove to be the most useful; it provides a means by which a basic movement such as grasping an object from one place on a table can be easily modified to grasping an object anywhere else on the table. This allows simple and straightforward modification of motion-capture data through a standard keyframing interface. Timewarping as a non-linear method to speed up or slow down motion is useful in blending different movements. It could also play an important role in synchronizing various movements in an animation as well as in synchronizing animation with sound. Multiresolution filtering has been demonstrated as an easy tool to change the quality of a motion. Waveshaping represents a simple but efficient way to introduce subtle effects to all or some degrees of freedom. As the use of motion capture is becoming increasingly popular and libraries of motions are increasingly available, providing alternate methods for modifying and tweaking movement for reuse can be of great value to animators.

We believe that a wide range of animation tasks can be addressed with these techniques at a high level which is complimentary to and extends conventional spline tweaking tools:

- blending of motions is straightforward by using multitarget interpolation with automatic time registration of movements. This is a convenient way to build up more complex motions from elementary ones. For more fine-control, the frequency bands can first be computed for each motion before blending band-by-band while adjusting the frequency gains.
- concatenating motions is another practical application of multitarget interpolation, giving the user control over blending interval (transition zone) and blending coefficient. Multiresolution can be applied to concatenate band-by-band.

- capping of joint angles is a task easily accomplished by wave-shaping. This tool is also well suited to apply user-defined undulations to all or some degrees of freedom to make a "bland" motion more expressive.
- some animation tasks which can be achieved with multiresolution analysis include "toning down" a motion by increasing the low frequency gains, "exaggerating" a movement by increasing the middle frequencies, producing a "nervous twitch" by increasing the higher frequencies, and generating "anticipation and follow-through" by assigning negative gain values. Because of immediate feedback, the user can quickly experiment with different combinations of gain values for specific movement qualities.
- editing of motion-captured data is very desirable yet very tedious in current systems. As mentioned above, displacement mapping provides an interface through which the animator can conveniently change such data at a few selected "keyframes" while preserving the distinctive "signature" of the captured motion.

References

- [1] BERGERON, P., AND LACHAPPELLE, P. Controlling facial expressions and body movements in the computer-generated animated short: Tony de Peltrie. In *Computer Graphics (SIGGRAPH '85), Course Notes: Techniques for Animating Characters* (July 1985).
- [2] BRUDERLIN, A., AND CALVERT, T. Interactive animation of personalized human locomotion. In *Graphics Interface '93, Proceedings* (May 1993), pp. 17–23.
- [3] BURT, P. Multiresolution method for image merging. In *Computer Graphics (SIGGRAPH '86), Course Notes: Advanced Image Processing* (August 1986).
- [4] BURT, P., AND ADELSON, E. A multiresolution spline with application to image merging. *ACM Transactions on Graphics* 2, 4 (October 1983), 217–236.
- [5] CALVERT, T., BRUDERLIN, A., DILL, J., SCHIPHORST, T., AND WELMAN, C. Desktop animation of multiple human figures. *IEEE Computer Graphics & Applications* 13,3 (1993), 18–26.
- [6] CASSELL, J. ET AL. Animated conversation: Rule-based generation of facial expression, gesture & spoken intonation for multiple conversational agents. In *Computer Graphics (SIGGRAPH '94 Proceedings)* (July 1994), pp. 413–420.
- [7] CHUI, C. K. *An Introduction to Wavelets, Series: Wavelet Analysis and its Applications*. Academic Press, Inc., 1992.
- [8] COHEN, M. Interactive spacetime control for animation. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), vol. 26, pp. 293–302.
- [9] DEMORI, R., AND PROBST, D. *Handbook of Pattern Recognition and Image Processing*. Academic Press, 1986, ch. Computer Recognition of Speech.
- [10] FUCHS, H., KEDEM, Z., AND USELTON, S. Optimal surface reconstruction from planar contours. *Communications of the ACM* 10, 10 (1977), 693–702.
- [11] GUO, S., ROBERGE, J., AND GRACE, T. Controlling movement using parametric frame space interpolation. In *Computer Animation '93, Proceedings* (1993), pp. 216–227.
- [12] ISAACS, P., AND COHEN, M. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (1987), vol. 21, pp. 215–224.
- [13] KASS, M. Condor: Constraint-based dataflow. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (1992), vol. 26, pp. 321–330.
- [14] KOCHANNEK, D., AND BARTELS, R. Interpolating splines with local tension, continuity and bias control. In *Computer Graphics (SIGGRAPH '84 Proceedings)* (1984), vol. 18, pp. 33–41.
- [15] KOGA, Y., KONDO, K., KUFFNER, J., AND LATOMBE, J.-C. Planning motions with intentions. In *Computer Graphics (SIGGRAPH '94 Proceedings)* (July 1994), pp. 395–408.
- [16] LEBRUN, M. Digital waveshaping synthesis. *Journal of the Audio Engineering Society* 27, 4 (1979), 250–266.
- [17] LITWINOWICZ, P. Inkwell: A 2 1/2-D animation system. In *Computer Graphics (SIGGRAPH '91 Proceedings)* (1991), vol. 25, pp. 113–122.
- [18] LIU, Z., GORTLER, S., AND COHEN, M. Hierarchical spacetime control. In *Computer Graphics (SIGGRAPH '94 Proceedings)* (July 1994), pp. 35–42.
- [19] MEYER, K., APPLEWHITE, H., AND BIOCCA, F. A survey of position trackers. *Presence: Teleoperators and Virtual Environments* 1, 2 (Spring 1992), 173–200.
- [20] ODGEN, J., ADELSON, E., BERGEB, J., AND BURT, P. Pyramid-based computer graphics. *RCA Engineer* 30, 5 (1985), 4–15.
- [21] PARKE, F. ET AL. State of the art in facial animation. In *Computer Graphics (SIGGRAPH '90), Course Notes* (August 1990).
- [22] PHILLIPS, C., AND BADLER, N. Interactive behaviors for bipedal articulated figures. In *Computer Graphics (SIGGRAPH '91 Proceedings)* (1991), vol. 25, pp. 359–362.
- [23] RASURE, J., AND KUBICA, S. The Khoros application development environment. Tech. rep., Khoros Research, Inc., 4212 Courtney NE, Albuquerque, NM, 87108, USA, 1993.
- [24] SEDERBERG, T., AND GREENWOOD, E. A physically-based approach to 2-D shape blending. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (1992), vol. 26, pp. 26–34.
- [25] SERRA, B., AND BERTHOLD, M. Subpixel contour matching using continuous dynamic programming. *IEEE Computer Vision and Pattern Recognition* (1994), 202–207.
- [26] STEIN, C., AND HITCHNER, H. The multiresolution dissolve. *SMPTE Journal* (December 1988), 977–984.
- [27] STURMAN, D. A discussion on the development of motion control systems. In *Graphics Interface '86, Tutorial on Computer Animation* (1986).
- [28] UNUMA, M., AND TAKEUCHI, R. Generation of human motion with emotion. In *Computer Animation '93, Proceedings* (1993), pp. 77–88.
- [29] VELHO, L. *Piecewise Descriptions of Implicit Surfaces and Solids*. PhD thesis, University of Toronto, Computer Science, 1994.
- [30] WILHELMS, J. Virya—a motion control editor for kinematic and dynamic animation. In *Graphics Interface '86, Proceedings* (1986), pp. 141–146.
- [31] WITKIN, A., AND KASS, M. Spacetime constraints. In *Computer Graphics (SIGGRAPH '88 Proceedings)* (1988), vol. 22, pp. 159–168.
- [32] WITKIN, A., TERZOPOULOS, D., AND KASS, M. Signal matching through scale space. *International Journal of Computer Vision* (1987), 133–144.
- [33] ZELTZER, D. Towards an integrated view of 3-D computer character animation. In *Graphics Interface '85, Proceedings* (1985), pp. 105–115.

Emotion from Motion

Kenji Amaya

Dept. of Mechanical and Environmental Informatics
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo 152, Japan
email kamaya@mes.titech.ac.jp

Armin Bruderlin

AIR Media Integration & Communication
Seika-cho Soraku-gun
Kyoto 619-02, Japan
email armin@mic.air.co.jp

Tom Calvert

Simon Fraser University
Burnaby, B.C. V5A 1S6, Canada
email tom@cs.sfu.ca
phone (604) 291-4588 fax (604) 291-4121

Abstract

This paper introduces a model to generate "emotional" animation from "neutral" human motion. Using techniques from signal processing, our method calculates certain *emotional transforms* which are then applied to existing motions of articulated figures in order to produce the same motions, but with an emotional quality such as angry or sad. These transforms capture the difference between a neutral and emotional movement with respect to two components: *speed* (timing), and *spatial amplitude* (range) of a movement.

Since the transforms are applied as global operations, they provide a convenient and efficient way to adapt motion captured, simulated or keyframed animation of articulated figures to different situations and characters.

Keywords: human figure animation, motion capture, motion control, digital signal processing

Introduction

In recent years, human animation has played an increasing role in such areas as advertising, entertainment, education, scientific visualization and simulation. However while many motion generation methods have been published, human animation is still in its infancy especially in the representation of expression, personality and emotion when compared to real human movement. Much of the difficulty in animating human motion can be attributed to the

many degrees of freedom that must be controlled even for simplified models. Another challenge in animating human movement is the fact that humans are very sensitive observers of each others' motion, in the sense that we can easily detect erroneous movement ("it simply doesn't look right"), although we often find it much more difficult to isolate the factor which causes the movement to look incorrect.

Motion capture techniques have come to the rescue since they preserve the distinctive "signature" of the real movement. However, motion capture has the disadvantage that special equipment is required and current systems allow for only limited editing capabilities to adapt a movement once it is captured, this requires the whole data capture process to be repeated if a motion sequence slightly different from an already captured one is desired.

In this paper, a method to produce emotional animation from neutral expressionless motion is proposed. This method can be divided into two parts: identification of certain *emotional transforms* by signal processing techniques and application to a "neutral" human motion to generate the same movement, but with an emotional trait. For example, if we apply the "angry" transform I_A to an animated sequence of a person drinking from a cup, the result will be an angry person drinking.

Our approach modifies existing animation data of articulated figures and therefore makes the use of motion capture, procedural, physically based and keyframe techniques more meaningful and useful. By

applying elementary techniques from signal processing, a high level interface to producing emotional animation is achieved. This approach is related to several other research efforts. Unuma et al [1, 2] apply Fourier transformations to data on human walking for animation purposes. Through Fourier expansions of the joint angles, a basic 'walking' factor and a 'qualitative' factor like "brisk" or "fast" are extracted. These factors are then used to generate new movements by interpolation and extrapolation in the frequency domain, such that now a walk can be changed continuously from normal to brisk walking, or a walk can be changed smoothly into a run. Litwinowicz uses recursive filters to produce "lag, drag, and wiggle" effects to keyframed two dimensional animated motion in a system called Inkwell [4]. Bruderlin and Williams [5] introduce a number of signal processing techniques for human figure animation which support various animation effects such as applying multiresolution filtering to exaggerate a movement, or automatically aligning two movements in time via non linear timewarping. They also apply displacement mapping to conveniently edit motion captured data. A similar technique called motion warping was proposed by Witkin and Popovic [6].

For the purpose of animating "emotions"¹, an emotion is considered to be a kind of *secondary* movement which piggybacks on top of a *primary* movement (see [10] for a discussion on primary/secondary movement, and [11] for a categorization for gestures). In this paper, we are mainly concerned with body movement, neglecting facial expression and speech which are also important factors in convincingly animating emotions in human characters. However, as explained in the next section, our technique is general in the sense that no matter what emotional motion is provided in calculating the emotional transform, the "difference" between the emotional and neutral motion is convincingly applied to a new, neutral movement.

In Section , our approach to generate emotional animation is described in more detail. Section discusses how the emotional transforms are derived and

¹Whereas the study of human emotions has led to some consensus over a definition about what constitutes an emotion — environmental and psychological events influence brain processes that actively modulate clearly observable behaviors [7] — several models exist on how to classify emotions into anger/sadness/happiness/fear/etc [7, 8]. Research also indicates that there are significant cultural differences in how emotions are expressed and perceived [9].

applied to new neutral movement. Results and examples are given in Section , and conclusions and future work are addressed in Section .

Basic Approach

Our technique to animate human emotions involves the following steps:

- 1 capture the motion of human subjects performed with different emotions, such as angry, sad, neutral,
- 2 for each emotion, calculate an *emotional transform* which is the "difference" between the neutral and emotional movement,
- 3 apply this emotional transform to a new, neutral movement.

We used an optical motion capture system (OPTOTRAK [12]) to record the movements as shown in Figure 1. In an experimental setup, subjects were asked to perform two motions with different emotions as well as in a neutral manner: "pick up the glass of water, drink from it, and put it back onto the table", and "knock at the door three times". A script was presented to each subject to provide situational context to each emotion². Each emotion was recorded three times. Six infrared emitting diodes (IRED's) were attached to the subjects' head, shoulder, elbow, wrist and hand. The system tracked the locations of each IRED³ by three calibrated cameras while calculating the absolute positions in space over time. From the six positions of the IRED's, nine rotational degrees of freedom (joint angles) were calculated in order to animate a human figure: two for sternum, three for the shoulder, one for the elbow and three for the wrist.

Steps two and three above address the main focus of this paper: how to abstract and represent the difference between a neutral and an emotional captured motion, so that it can be applied to a new movement to make it emotional. After careful analysis of the motion captured data, we identified two components which vary noticeably over the various emotions: *speed* (timing) and *spatial amplitude* (range) of the motion. Figure 2 illustrates these variations in

²A total of ten emotions or moods were captured — neutral, angry, sad, happy, fearful, tired, strong, weak, excited, and relaxed — although for the analysis here we concentrate on just neutral, angry, and sad.

³A sampling rate of 120 Hz was chosen; accuracy was within 0.5mm.

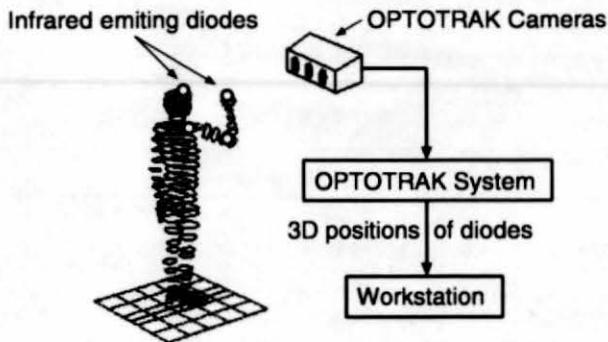


Figure 1: Motion capture system.

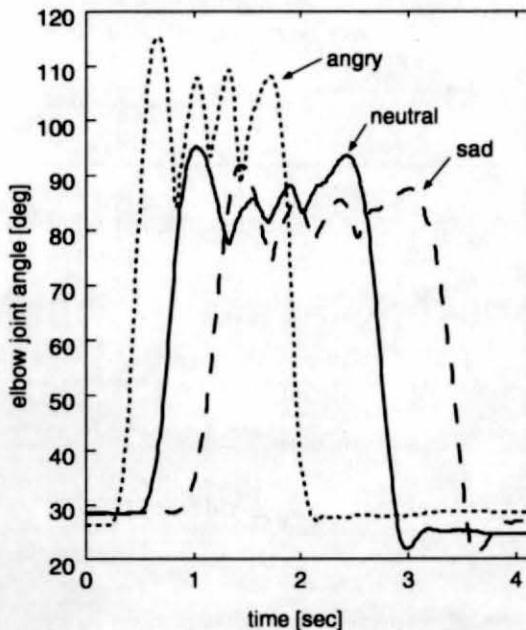


Figure 2: Knocking with different emotions.

the motion-captured data of the knocking movement; the joint angles of the elbow show significant differences in time and amplitude for the neutral, angry and sad motion, respectively.

As we explain in the next section, the *speed* transform has been implemented as a non-linear time-warping technique, and the *spatial amplitude* transform is based on signal amplifying methods. Applying these transforms produces an emotional human movement from a new, neutral movement. To verify our method, we applied the following procedure, examples of which are presented in section :

1. calculate the angry and sad transforms from captured drinking data.

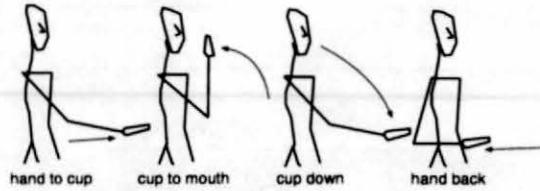


Figure 3: Basic periods for drinking motion.

2. apply the angry and sad transforms to neutral, captured drinking data and compare with captured angry and sad data.
3. apply the angry and sad transforms (calculated from drinking data) to neutral knocking data and compare with captured angry and sad knocking data.

Algorithm

This section focuses on how the transforms for speed (section) and spatial amplitude () are derived from existing motion-captured data, and then applied to a new neutral movement. In order to provide a general technique which works with different motion data, we first subdivide both the neutral and emotional captured data into units of motion, called "basic periods". These periods are bounded by the time when the velocity of the wrist change its direction, or they lie between the extrema which separate extension and flexion in a joint ⁴. For example, for the drinking motion we used to derive the emotional transforms the basic periods are "hand to a cup, cup to mouth, cup down, hand back" as shown in Figure 3.

Transform of Speed

The first step to obtain the *speed* transform is to calculate the absolute speed of the end effector which is the wrist point in our case for the drinking motion (where speed is defined along the trajectory of the wrist) for both neutral and emotional motion data (see Figure 4).

After determining the basic periods as defined above, one of them is selected and integrated along the trajectory. For the drinking motion the period with the longest duration was selected ("cup down"). The calculated data can now be represented as:

$$s = f_N(t) = \int_0^t |\mathbf{v}_N(\tau)| d\tau ; \quad (1)$$

⁴The basic periods can also be specified directly by the user based on the joint angle or velocity trajectories.

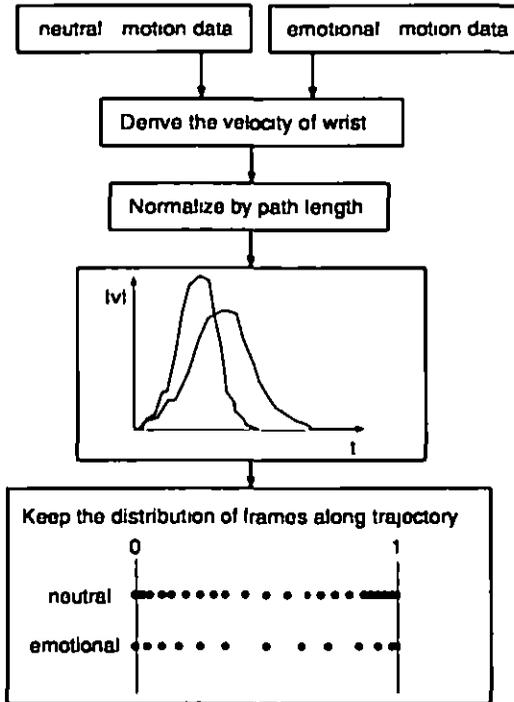


Figure 4 Algorithm to obtain speed transform

$$s = f_E(t) = \int_0^t |v_E(\tau)| d\tau \quad (2)$$

where t is the time, s the position along the trajectory, $v(t)$ the 3D velocity vector of the wrist and the subscript N and E denote "neutral" and "emotional", respectively. These data are normalized along the trajectory as follows, where t_{end} is the duration of the basic period

$$s = f_{N \rightarrow E}(t) = f_{N \rightarrow E}(t) / \int_0^{t_{end}} |v_{N \rightarrow E}(\tau)| d\tau \quad (3)$$

The distribution of frames $\rho(s)$ is calculated by

$$\rho_{N \rightarrow E}(s) = n \frac{dt}{ds} = \frac{n}{f'_{N \rightarrow E}(f^{-1}(s))} \quad (4)$$

where n is the number of frames per second, $\rho_N(s)$ and $\rho_E(s)$ are used as templates when applying the transform

The first step in applying the speed transform to a new, neutral movement is the calculation of the distribution of its frames, followed by a division into "basic periods" as illustrated in Figure 6

For each basic period, the following calculation steps are then performed as shown in Figure 5: normalize (scale) period in length, substitute the "emotional" distribution of the frames for the neutral one,

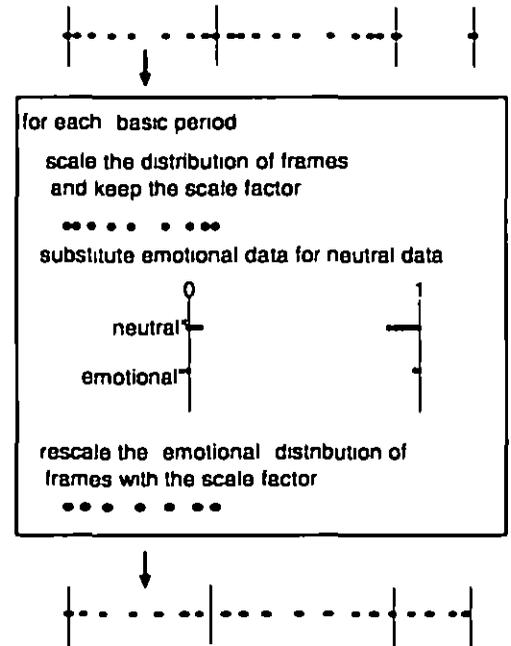


Figure 5 Application of speed transform

divide the new motion data into basic periods

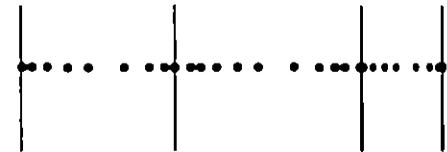


Figure 6 Keyframe distribution of new motion data

re-scale basic period (inverse of initial normalization)

This warped frame distribution is used as a correspondence table in the "emotional" joint angle calculations. Figure 7 gives an example: the emotional joint angle of the 5th frame is obtained by interpolating between the 6th and 7th frame in the original data, because the emotional 5th frame corresponds to the 6.8th frame in the original data.

Transform of Spatial Amplitude

The transform of spatial amplitude is obtained by applying the algorithm described below

for both neutral and angry motion, divide the joints into four categories corresponding to the levels of hierarchy of the articulated figure. This division is necessary because the range of motion for the joints are substantially different in each category. Figure 8 shows which joint belongs to which category. For

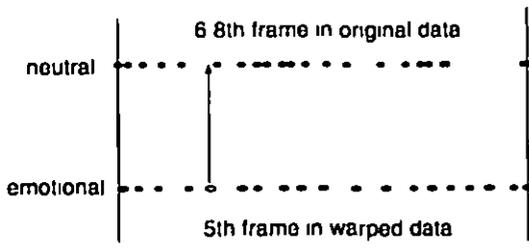


Figure 7 Generation of joint angle data

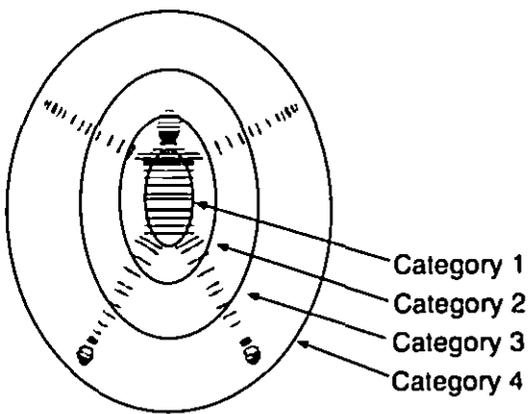


Figure 8 Joint categories

example, the sternum belongs to the category one, and a knee belongs to category three"

Each category represents a multi dimensional space which is defined by its joint angles and time t . Let us define these multi dimensional spaces by $\theta_{N_o E}^i(t)$, where i ($1 \leq i \leq 4$) denotes the corresponding category. For example, the multi dimensional space for category one has 35 degrees of freedom (34 joint angles plus time)

In order to extract the intensity of the spatial amplitude from both "neutral" and "emotional" motion data, the factor $d_{N_o E}^i$ is defined as follows for each basic period in turn

$$d_{N_o E}^i = \max(|\theta_{N_o E}^i(t) - \{\theta_{N_o E}^i(t_{init})(1-t) + \theta_{N_o E}^i(t_{end})t\}|), \quad (5)$$

where $|\cdot|$ is the Euclidean norm operation. This calculation can be represented conceptually as shown in Figure 9. A straight line is drawn from the initial point to the end point of the current basic period for each category i for both "neutral" and

⁵Category one involves 34 joints category two involves 26 joints category three involves four joints category four involves 12 joints in our model of the human figure

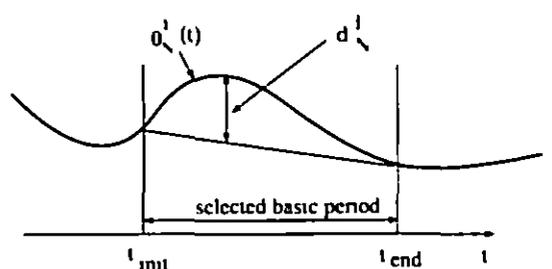


Figure 9 Intensity factor of spatial amplitude d^i

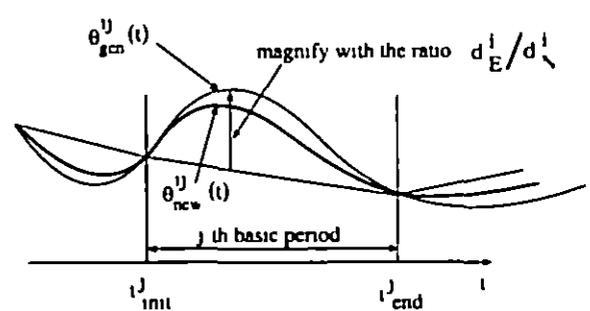


Figure 10 Transformation of spatial amplitude

"emotional" motion data. The maximum distances $d_{N_o E}^i$ between this straight line and the trajectory of motion are calculated in this space, and can be considered as intensities of spatial amplitude for each neutral and emotional motion.

The spatial amplitude transform is now applied to a new, neutral motion, $\theta_{org}^j(t)$, where j is the corresponding basic period. This results in a new, emotional motion, $\theta_{gcn}^j(t)$, defined by

$$\theta_{gcn}^j(t) = \frac{d_F^j}{d_N^j} \theta_{org}^j(t) + \frac{d_N^j - d_F^j}{d_N^j} \{\theta_{N_o E}^j(t_{init}^j)(1-t) + \theta_{N_o E}^j(t_{end}^j)t\} \quad (6)$$

The basic idea of this amplitude transform is nonlinear magnification (see Figure 10), draw a straight line from the initial point to the end point of each basic period in the time joint angles space of each category. Magnification is performed on the distance between this straight line and the trajectory of motion by the ratio $\frac{d_E^i}{d_N^i}$ for each category i and each basic period j .

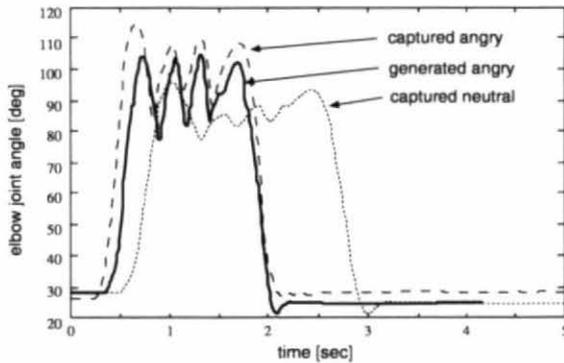


Figure 11: Comparison between generated and captured elbow joint-angles.

Results

Examples

After applying the angry and sad transforms derived from the motion captured drinking motion to the neutral, captured drinking data, we obtained a close match with the “real” (motion-captured) angry and sad drinking data. The same emotional transforms were then applied to the motion of knocking at a door. The resulting motions were compared to the motion-captured angry and sad knocking movements to verify the model. Figure 11 shows the elbow joint angle during a knocking movement; it is seen that the generated angry movement data is a good fit to the real captured data.

The same emotional transforms have also been applied to a plain, keyframed kicking motion which produced believable emotional variations of the motion. Figure 12 shows snapshots of this kicking animation.

High Frequencies

Besides the speed and amplitude transforms, we searched for other components in the joint angle signals which could produce a significant effect between neutral and emotional signals. Initially, the frequency content was also considered as an important aspect of movement. However, results of frequency analysis revealed that high frequency components in the data are neglectable in both the “emotional” and “neutral” signals. Figure 13 shows the power spectrum of the wrist position; we can see that the signals do not have major components above 10 Hz.

Thus, high frequencies in human motion can be ignored for the purposes of animation where, in any event, a sampling rate of 30 frames/sec or less does



animated “neutral” kicking motion;



generated “sad” kicking motion;



generated “angry” kicking motion;

Figure 12: Keyframes for animated kicking motion.

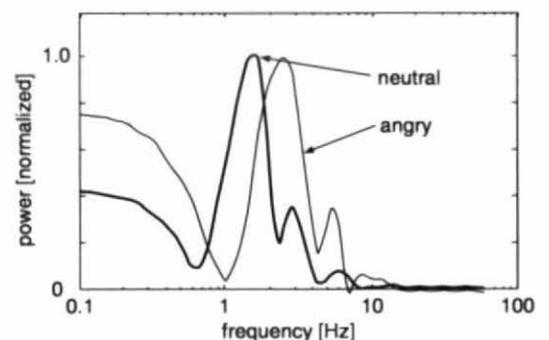


Figure 13: Power spectrum of wrist position data.

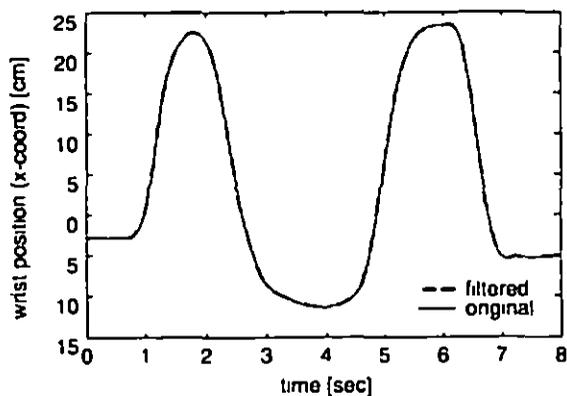


Figure 14 Comparison between low pass filtered data and original data

not allow for accurate reproduction of higher frequency components. Furthermore, what we can not see in animation is not important. Figure 14 gives a comparison between low pass filtered data (applying a Gaussian filter kernel of width five) and original data plotted against time, there is no noticeable difference between the two signals.

Phase Shift

Rather than high frequencies which are often regarded as providing the realistic "signature" in human animation, we believe that movement *phase shift* is a distinctive feature of real human motion. Phase shift is the amount with which the movements of the individual joints overlap⁶. Figures 15 and 16 show the joint angle velocity over time for the neutral and angry drinking data, where each signal is normalized by its maximum value. The maximum velocity timings show this phase shift clearly. Moreover, we think that these phase shifts are different between "neutral" and "angry" motion. More research will be necessary to determine how such a phase shift transform could be incorporated into our algorithm.

Conclusions and Future work

A model has been developed to produce "emotional" animations from "neutral" human motion. The method is based on signal processing techniques which analyze experimental data of emotional human motion and extract the difference between emotional and neutral movement. Two components are isolated to define this difference, a *speed* and a *partial amplitude* transform. These two components are

⁶This has long been regarded as an important principle in articulated figure animation [3].

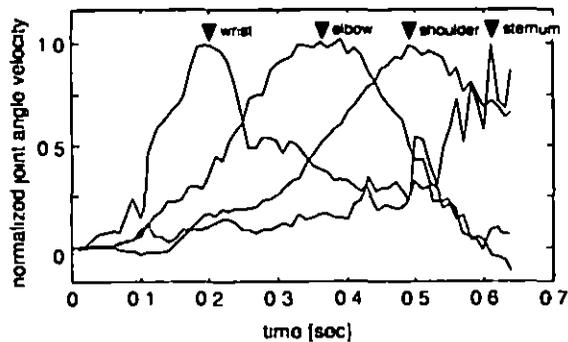


Figure 15 Neutral joint angles

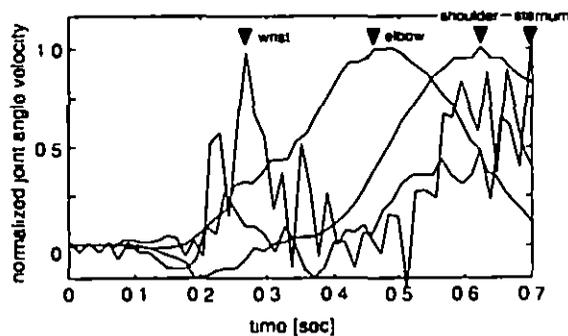


Figure 16 Angry joint angles

then applied to new, neutral movements to generate "emotional" movements. In order to verify our approach, angry and sad transforms from a motion captured drinking sequence have been applied to a neutral knocking motion. The calculated angry and sad knocking motions came very close to the real, captured angry and sad knocking motions. By establishing various categories for degrees of freedom of the human figure, this method is general with respect to employing an emotional transform derived, say, for the arm to a motion of another body part. Emotional kicking motions were generated by applying the drinking transform to the lower body.

By automating the generation of emotional animations, our technique facilitates the reuse and adaptation of existing motions of articulated figures and makes the use of motion libraries of keyframed, motion captured, simulated or procedurally generated movements more meaningful. Also, since the computations for deriving the emotional transforms are done off line, this approach could well be useful in changing the emotions of virtual actors on the fly in real time environments.

We are currently investigating how this approach

can be extended in various ways. Besides emotions, it is desirable to generate animations with different personalities, cultures and genders. Also, we hope to capture the difference in motion between an old man and a young boy in a similar way. Furthermore, using the same emotional transforms derived from human walking data to produce an angry walking sequence of a dog, for instance, from the neutral motion would be a practical generalization of our method.

References

- [1] Munetoshi Unuma and Ryozo Takeuchi, "Generation of human motion with emotion", in *Computer Animation 93 Proceedings*, 1993, pp 77-88
- [2] Munetoshi Unuma, Ken Anjou, and Ryuzo Takeuchi, "Fourier principles for emotion based human figure animation", in *Computer Graphics (SIGGRAPH 95 Proceedings Los Angeles CA August 6 11)*, 1995, pp 91-96
- [3] John Lasseter, "Principles of traditional animation applied to 3-D computer animation", in *Computer Graphics (SIGGRAPH 87 Proceedings)*, 1987, vol 21, pp 35-44
- [4] Peter Litwinowicz, "Inkwell: A 2 1/2-D animation system", in *Computer Graphics (SIGGRAPH 91 Proceedings)*, 1991, vol 25, pp 113-122
- [5] Armin Bruderlin and Lance Williams, "Motion signal processing", in *Computer Graphics (SIGGRAPH 95 Proceedings Los Angeles CA August 6 11)*, 1995, pp 97-104
- [6] Andrew Witkin and Zoran Popovic, "Motion warping", in *Computer Graphics (SIGGRAPH 95 Proceedings Los Angeles CA August 6 11)*, 1995, pp 105-108
- [7] Robert Plutchick and Henry Kellersman, Eds, *Emotion: Theory, Research and Experience, Volume 1-3*, Academic Press, 1986
- [8] Carroll E. Izard, Ed, *The Psychology of Emotions*, Plenum Press, New York, 1991
- [9] Shunya Sogon and Makoto Masutani, "Identification of emotion from body movements: A cross cultural study of Americans and Japanese", *Psychological Reports*, , no 65, pp 35-46, 1989
- [10] Claudia Morawetz, "A high level approach to the animation of human secondary movement", Master's thesis, Simon Fraser University, School of Computing Science, April 1989
- [11] Ray L. Birdwhistell, *Kinetics and Context: Essays on Body Movement Communication*, University of Pennsylvania Press, Philadelphia, 1970
- [12] Northern Digital Inc, *Northern Digital Op tottrak Manual and Product Information*, J R Krist, 403 Albert Street Waterloo, Ontario N2L 3V2 Canada, October 1991

IMPROV

New Developments in Improvisational Animation at the Media Research Lab

**Ken Perlin/Athomas Goldberg
NYU Media Research Lab
719 Broadway 12th floor
New York, NY 10003
[http //www mrl nyu edu/improv](http://www.mrl.nyu.edu/improv)**

Introduction

The use of interactive animated agents - lifelike representations of real and imaginary entities - in virtual environments has become an interesting and important field of research as the technologies for creating these environments have become progressively more inexpensive and widespread. Until recently the computer graphics hardware necessary for producing real-time 3D simulations was so expensive as to prohibit its use outside of military and industrial applications. Now, as consumer level PC's become increasingly powerful, and off-the-shelf 3D graphics support becomes more readily available, the application of these technologies to educational and other purposes has become feasible. With this comes a new set of demands. Military and industrial simulations have often viewed the use of animated agents in purely functional terms. The use of animated agents for human factors research in vehicle and factory design has become widespread, but these applications have generally focused on the mechanics of human movement [Badler91]. The military has made extensive use of animated agents in battlefield simulations but in this case agent behavior has more often than not been restricted to troop movement and general infantry behavior (run, shoot, etc.). The requirements for display of social and emotional behavior of animated agents in these simulations have been relatively limited.

Animated agents are now finding their way into numerous other development efforts and applications. Cognitive scientists and artificial intelligence researchers are using animated agents to expand their investigations into the physical expression of human behavior as it relates to the workings of the human mind [Hayes-Roth96]. Educators, struggling with the limitations of overcrowded classrooms and limited teacher time and resources, are looking to animated agents and virtual environments as a means for providing personalized guidance and tutoring to students. Historians and social scientists now have many of the tools needed to create lifelike, interactive simulations of situations and events which once could only be conveyed through static presentations of collected research and data. Beyond the realm of research and education, animated agents can put a human face on digital assistants and proxies in collaborative virtual environments for such tasks as communication and negotiation in the absence of, and in support of the human participants. In such contexts, participants must be able to interact with agents in meaningful ways and must feel that the agent with which they are interacting adequately represents the agent's principal.

Two basic approaches are currently combined for creation of animated agents: physical and bio-mechanical simulation and machine learning/artificial intelligence systems. Physical and bio-mechanical simulation systems are used to construct accurate models of human movement and constraints and apply them to situations involving physical activities. These systems use these models to

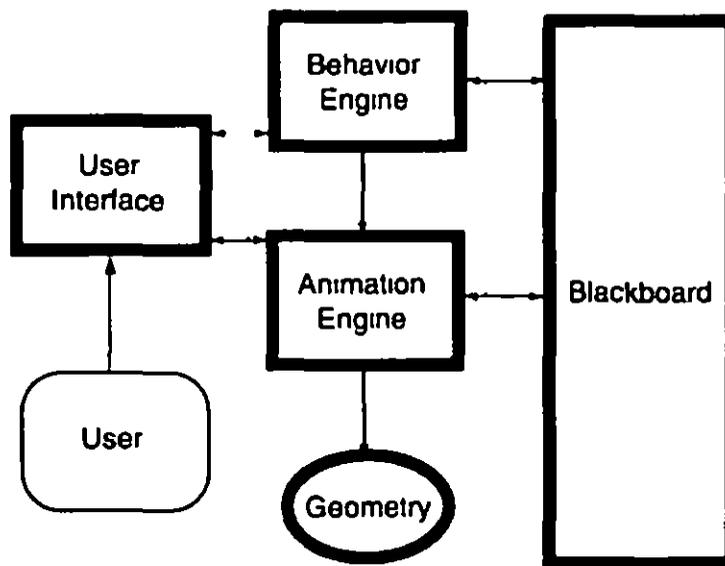
represent a range of physical tasks, from walking, to operating machinery, to negotiating complex terrain in a manner consistent with the capabilities and limitations of human movement. Machine learning and artificial intelligence systems generally seek to simulate human problem solving, in order to create agents who, at some level, think like human beings. They use means ranging from language parsing systems which apply rules for interpreting user entered text (or speech) in order to determine explicit and implicit meaning, to neural nets and fuzzy systems which attempt to develop behavioral rules based on analysis of changing situations and data over time.

Each of these approaches addresses a significant aspect of the animated agent problem. Physical and bio-mechanical models provide realistic and believable representations of human movement through a wide range of changing physical conditions within the virtual environment while artificial intelligence systems can allow the system to analyze and interpret complex user input and data. Together they address a large part of the animated agent problem, but there is still one significant piece of the puzzle missing. Neither of these approaches makes it easy enough to model agent 'mood' or personality, or the effect of these internal parameters on the way that an agent responds to events. However, it is just these complex and subtle aspects of human character that define what we perceive as individuality. Without this, the most sophisticated physical models and machine learning tools can only produce simulations which, though they may represent certain relatively mechanical aspects of human capability adequately, are, ultimately, devoid of life. Our work addresses this aspect of the interactive simulation of human behavior.

Conveying mood and personality is crucial to many new applications in which animated agents are used to express emotional messages or portray specific characters rather than just to solve an abstract problem. In such applications, the way in which the agent presents information can be as important as the information being presented.

The focus of the Improv project at NYU has been to provide tools which make it possible to create applications involving animated agents that behave, interact and respond to user input in ways that convey mood and emotion. These tools can be used without prior experience in computer programming, cognitive science or ergonomic simulation while still allowing creation of animated agents who exhibit behavior which has the lifelike, somewhat unpredictable feel of human behavior, yet remains consistent with a character's personality and defined goals.

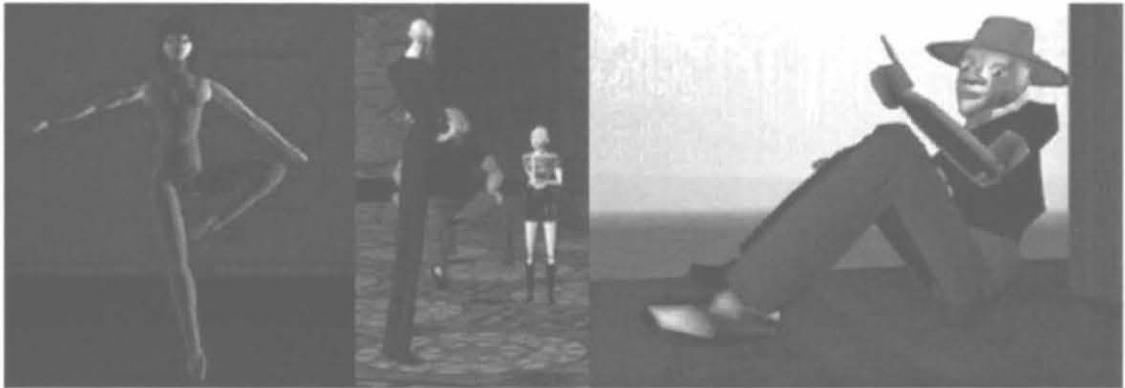
Improv employs a variety of techniques to accomplish this. Human beings are capable of performing many different activities simultaneously, while internally maintaining long-term goals and motivations. For animated agents to appear lifelike and believable, they must mimic these internal structures. To allow this, Improv is structured around an engine which continuously maintains and updates the state of the animated agent. This provides a behavioral layering mechanism which enables authors to create multiple behaviors that run simultaneously, each of which may activate or influence other behaviors. Layers of motivation can manifest themselves in activities carried out simultaneously such as walking or running combined with changes in facial expression. These layers are structured hierarchically so that behaviors on a layer responsible for maintaining long term goals can activate behaviors on layers containing the activities needed to carry out those goals, and so on. Improv provides authors with the ability to create such behaviors within a continuum that ranges from linear sequencing where each action comprising a behavior is carried out a specific time and in a predetermined order, to stochastic selection according to author-defined rules. Such rules are defined using a set of tools which enable the author to control the way in which user input and environment-derived clues influence on animated agent's apparent mood, personality and behavior. Also, because coordinated action of a group of separate graphical agents is often crucial, Improv allows the author to define layers of behavior and behavioral rules which apply to the activities of entire agent groups, while still providing the individual agent with its own unique personality.



A graphical agent has both a behavior engine and an animation engine agents communicate with each other through a blackboard protocol

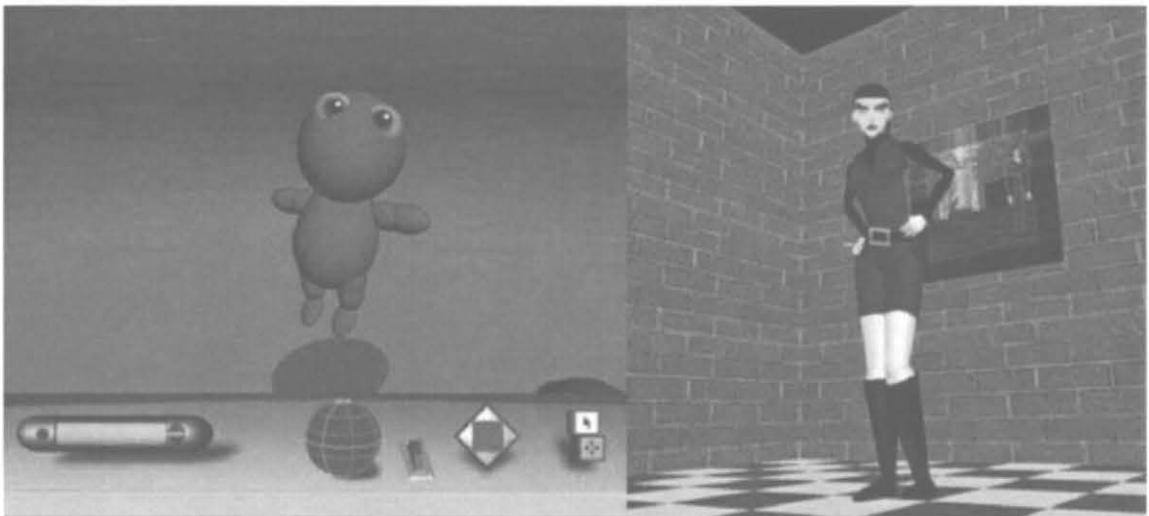
This approach combines concepts adapted from various fields of study. The Improv engine is roughly based on the Subsumption Architecture approach of [Brooks86] which has been used for numerous embodied agent systems including [Blumberg95]. Although most of these systems enforce a strict top down hierarchy of behavior, Improv's model is heterarchical, enabling authors to define their own connections between various layers of behavior and how these layers interrelate. Improv employs statistically controlled randomness to let authors create behavioral tendencies that are unpredictable yet consistent with an agent's character and personality. The methods employed to achieve this are conceptually similar to techniques employed in neural nets and fuzzy systems. However, rather than automatically generating statistics based on an analysis of large quantities of data, authors shape statistical parameters based on their concept of the agent's goals and motivations. Perhaps the most crucial inspiration for this work comes out of improvisational theater, where actors make use of numerous techniques to portray a wide variety of characters and scenarios in an environment in which events are not predetermined and where outcomes are subject to numerous influences including audience response and interaction.

The Improv Project is an ongoing research effort. A layered animation engine based on these principals was first described in [Perlin95] after being demonstrated in "Danse Interactif" [Perlin94] at the SIGGRAPH94 Electronic Theater. Linear and non-linear control over sequences of behavior and of explicit, random and rule based behavior activation was demonstrated at the SIGGRAPH95 Interactive Entertainment exhibit and at SIGGRAPH96 where it was formally presented in [Perlin96] and demonstrated in the *Digital Bayou* exhibit. These demonstrations featured simple examples involving small groups of agents engaged in specific activities for limited periods of time. These examples were developed using a programming interface which, while quite useable, restricted the use of Improv to authors with considerable programming experience and knowledge of the inner workings of the system.



*Scenes from "Danse Interactif", SIGGRAPH '94,
"Interacting with Virtual Actors", SIGGRAPH '95,
and "The Botanica Virtùal", SIGGRAPH '96*

Since then, Improv has been re-implemented in *Java* to allow wide-scale deployment over the World Wide Web. Its architecture has been redesigned to be more object-oriented and to allow for extensibility of the system by independent developers through a common API. This work was demonstrated at SIGGRAPH '97 during the course "Virtual Humans: Behaviors and Physics, Acting and Reacting" In addition to the basic behavior layering and control mechanisms, a set of animation tools has been provided for creating simple animated actions which can be blended together and composited, in much the same way that images are composited in *Photoshop*, along with a set of simple graphic user interface tools for editing these animations. As a result of this effort, running examples and an authoring kit are now freely available to other researchers at <http://mrl.nyu.edu/improv>, runnable within standard Web browsers on either PC or Silicon Graphics clients.



*Sid (left) and Wendy (right)
Created using Java version of Improv running within a VRML browser*

Relation to Other Work in Progress

Several ongoing research efforts at the NYU Media Research Lab are directly or indirectly related to the work being done in IMPROV.

Our laboratory is doing work on miniature direct-drive actuators, which we are using for self-propelling autonomous legged miniature robots. We are using control algorithms from Improv to create a sense of

life-like movement to these robots

Improv is also providing content for our work in active autostereoscopic displays. This research tracks the user's left and right eye positions at interactive rates and projects a different view onto each eye using a novel projection masking technique developed in our laboratory. Because Improv characters can dynamically modify body orientation and gaze direction we are using them as test subjects within responsive virtual autostereoscopic scenes.

We are currently working on the *Pad* multiscale interface. The emerging GUI of Improv provides a context for our work in creating recursive multiscale interface controllers. These allow authors to create arbitrarily nested behavior logic modules. The demands of the Improv GUI design are driving a number of innovations in our research on *Pad* zoomable GUI design and implementation.

Relation to Present State of Knowledge in the Field

Our planned work is related to other research efforts which have inspired or informed it and provide alternatives to or complements to the technologies we are developing in IMPROV. These include

The *Desktop Theater* was conceived by Steve Strassman [Strassman91]. He emphasized the importance of expressive authoring tools for specifying how characters would respond to direction. His goals were quite similar to ours. Yet because his work predated the age of fast graphical workstations it did not include real time visual interaction.

Most autonomous actor simulation systems follow the parallel layered intelligence model of [Minsky86], which was partially implemented by the subsumption architecture of [Brooks86] as well as in [Bates92] and [Johnson94]. Several systems have been developed which share this layered architecture with Improv, yet which solve distinctly different problems. The Jack system of [Badler93] focuses on proper task planning and biomechanical simulation, as does [Hodgins95]. Its general goal is to produce accurate simulations of biomechanical robots. Similarly the simulations of Terzopoulos et al [Terzopoulos94] has simulated autonomous animal behaviors that respond to their environment according to biomechanical rules. Autonomous figure animation has been studied by [Badler91], [Girard85], [Morawetz90] and [Sims94].

The *Alive* system of [Mac95] and [Blumberg95] focuses on self-organizing embodied agents which are capable of making inferences and of learning from their experiences. Instead of maximizing an author's ability to express personality the *Alive* system uses ethological mechanisms to maximize the actor's ability to reorganize its own personality, based on its own perception and accumulated experience.

The novel *Snow Crash* by Neil Stephenson also influenced this work. That novel posits a *Metaverse*, a future version of the Internet which appears to its participants as a quasi-physical world. Participants are represented by fully articulate human figures, or avatars. Body movements of avatars are computed automatically by the system.

References

[Badler91] N. Badler, B. Barsky, D. Zeltzer, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[Badler93] N. Badler, C. Phillips, B. Webber, *Simulating Humans: Computer Graphics, Animation, and Control*. Oxford University Press, 1993.

- [Bates92] J Bates A Loyall W Reilly, *Integrating Reactivity, Goals and Emotions in a Broad Agent* Proceedings of the 14th Annual Conference of the Cognitive Science Society, Indiana July 1992
- [Blumberg95] B Blumberg, T Galyean, *Multi Level Direction of Autonomous Creatures for Real Time Virtual Environments* Computer Graphics (SIGGRAPH '95 Proceedings), 30(3) 47--54 1995
- [Broderlin95] A Broderlin L Williams *Motion Signal Processing* Computer Graphics (SIGGRAPH '95 Proceedings), 30(3) 97--104 1995
- [Brooks86] R Brooks *A Robust Layered Control for a Mobile Robot* IEEE Journal of Robotics and Automation, 2(1) 14--23, 1986
- [Chadwick89] J Chadwick D Haumann R Parent, *Layered construction for deformable animated characters* Computer Graphics (SIGGRAPH '89 Proceedings), 23(3) 243--252, 1989
- [Ebert94] D Ebert and et al , *Texturing and Modeling, A Procedural Approach* Academic Press, London, 1994
- [Girard85] M Girard, A Maciejewski, *Computational modeling for the computer animation of legged figures* Computer Graphics (SIGGRAPH '85 Proceedings), 20(3) 263 270, 1985
- [Hayes-Roth96] Hayes-Roth, B and van Gent R *Improvisational puppets actors and avatars* In Proceedings of the Computer Game Developers' Conference, Santa Clara CA, 1996
- [Hodgins95] J Hodgins W Wooten, D Brogan, J O'Brien, *Animating Human Athletics* Computer Graphics (SIGGRAPH '95 Proceedings), 30(3) 71 -78, 1995
- [Johnson94] M Johnson *WavesWorld A Testbed for Three Dimensional Semi-Autonomous Animated Characters (PhD Thesis)* MIT 1994
- [Johnson93] *Illusion of Life*
- [Maes95] P Maes, T Darrell and B Blumberg, *The Alive System Full Body Interaction with Autonomous Agents in Computer Animation* Animation '95 Conference, Switzerland April 1995 IEEE Press, pages 11-18
- [Minsky86] M Minsky, *Society of Mind* MIT press, 1986
- [Morawetz90] C Morawetz, T Calvert, *Goal directed human animation of multiple movements* Proc Graphics Interface } pages 60--67 1990
- [Perlin85] K Perlin, *An image synthesizer* Computer Graphics (SIGGRAPH '85 Proceedings) } 19(3) 287--293, 1985
- [Perlin94] K Perlin, *Danse interactif* SIGGRAPH '94 Electronic Theatre, Orlando
- [Perlin95] K Perlin, *Real Time Responsive Animation with Personality* IEEE Transactions on Visualization and Computer Graphics, 1(1), 1995

- [Perlin96] K Perlin, A Goldberg *Improv: A System for Scripting Interactive Actors in Virtual Worlds* Computer Graphics, Vol 29 No 3 (online at <http://www.mrl.nyu.edu/improv>)
- [Sims94] K Sims *Evolving virtual creatures* Computer Graphics (SIGGRAPH 94 Proceedings) 28(3) 15--22 1994
- [Stephenson92] N Stephenson *Snow Crash* Bantam Doubleday, New York, 1992
- [Strassman91] S Strassman, *Desktop Theater: Automatic Generation of Expressive Animation* (PhD thesis) MIT Media Lab June 1991 (online at http://www.method.com/straz/straz_phd.pdf)
- [Terzopoulos94] D Terzopoulos, X Tu, and R Grzeszczuk, *Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World* Artificial Life 1(4) 327-351, 1994
- [Witkin95] A Witkin, Z Popovic, *Motion Warping* Computer Graphics (SIGGRAPH 95 Proceedings) 30(3) 105-108 1995

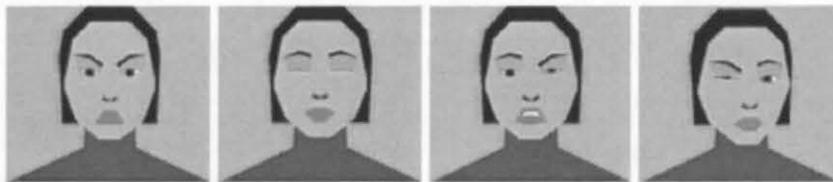
Layered Compositing of Facial Expression

SIGGRAPH 1997 technical sketch

Ken Perlin
Media Research Laboratory
Department of Computer Science
New York University

How does one make an embodied agent react with appropriate facial expression, without resorting to repetitive prebuilt animations? How does one mix and transition between facial expressions to visually represent shifting moods and attitudes? How can authors of these agents relate lower level facial movements to higher level moods and intentions? We introduce a computational engine which addresses these questions with a stratified approach. We first define a low level movement model having a discrete number of degrees of freedom. Animators can combine and layer these degrees of freedom to create elements of autonomous facial motion. Animators can then recursively build on this movement model to construct higher level models.

In this way, animators can synthesize successively higher levels of autonomous facial expressiveness. A key feature of our computational approach is that composited movements tend to blend and layer in natural ways in the run time system. As the animator builds at higher levels, the correct layering priorities are always maintained at lower supporting levels. We have used this approach to create emotionally expressive autonomous facial agents.



angry, daydreaming, disgusted, distrustful



fiendish, haughty, head back, scolding, sad



smiling, sneezing, surprised, suspicious

Building on our work in *Improvisational Animation* [Siggraph 96], we use a parallel layered approach. Our authoring system allows its user to relate lower level facial movements to higher level moods and

intentions by using a model inspired by optical compositing. We first allow the animator to abstract facial motion into a discrete number of degrees of freedom. The system does not impose a particular model at this stage, but rather allows the animator to define the degrees of freedom that he or she finds useful.

Given a set of degrees of freedom, we allow the animator to specify time-varying linear combinations and overlays of these degrees of freedom. Each definition becomes a new, derived degree of freedom. Collectively, these derived degrees of freedom create a new abstraction layer. We allow animators to recursively create successive abstractions, each built upon the degrees of freedom defined by previous ones.

More specifically, we internally represent each of the model's K degrees of freedom by a K -dimensional basis vector, which has a value of 1 in some dimension j and a value of 0 in all other dimensions. The state space for the face consists of all linear combinations of these basis vectors. We allow the animator to create time-varying functions (coherent noise and splined curves) of these linear combinations, to create a simple vocabulary of "basis motions." We then provide a motion compositing engine, which allows the animator to specify a compositing structure for these basis motions. The engine gives motions varying "opacities" and does alpha blending, much as Photoshop composites layers of images. The animator can then encapsulate sets of time-varying weights for each of the basis motions in the above engine, and define each such set as a new motion basis. By doing this recursively, the animator can describe successively higher levels of an emotional vocabulary.

We have found that in this way, successively higher semantic levels of facial expressiveness can be effectively synthesized. In particular, we find that movements defined with this method tend to blend together and overlay in natural ways. As the animator works within higher abstractions, the system always maintains correct priorities at all lower supporting abstractions. The result is real-time interactive facial animation that can achieve a convincing degree of emotive expressiveness.

The eventual goal of this work is to give computer/human interfaces the ability to represent the subtleties we take for granted in face-to-face communication, so that they can function as agents for an emotional point of view. By automating the creation of facial expression in the run-time engine of an interactive agent, we enable such an agent to operate without the explicit intervention of a human operator.

Improv A System for Scripting Interactive Actors in Virtual Worlds

Ken Perlin / Athomas Goldberg
Media Research Laboratory
Department of Computer Science
New York University
perlin@nyu.edu | athomas@mrl.nyu.edu

ABSTRACT

Improv is a system for the creation of real time behavior based animated actors. There have been several recent efforts to build network distributed autonomous agents. But in general these efforts do not focus on the author's view. To create rich interactive worlds inhabited by believable animated actors, authors need the proper tools. **Improv** provides tools to create actors that respond to users and to each other in real time, with personalities and moods consistent with the author's goals and intentions.

Improv consists of two subsystems. The first subsystem is an Animation Engine that uses procedural techniques to enable authors to create layered, continuous, non-repetitive motions and smooth transitions between them. The second subsystem is a Behavior Engine that enables authors to create sophisticated rules governing how actors communicate, change, and make decisions. The combined system provides an integrated set of tools for authoring the minds and bodies of interactive actors. The system uses an English style scripting language so that creative experts who are not primarily programmers can create powerful interactive applications.

INTRODUCTION

Believability And Interaction

Cinema is a medium that can suspend disbelief: the audience enjoys the psychological illusion that fictional characters have an internal life. When this is done properly, these characters can take the audience on a compelling emotional journey. Yet cinema is a linear medium: for any given film, the audience's journey is always the same. Likewise, the experience is inevitably a passive one as the audience's reactions can have no effect on the course of events.

This suspension of disbelief, or believability, does not require realism. For example, millions of people relate to Kermit the Frog and to Bugs Bunny as though they actually exist. Likewise, Bunraku puppet characters can create for their audience a deeply profound and moving psychological experience.

All of these media have one thing in common: Every moment of the audience's journey is being guided by talented experts, whether an screenwriter and actor/director, a writer/ animator, or a playwright and team of puppeteers. These experts use their judgment to maintain a balance: characters must be consistent and recognizable, and must respond to each other appropriately at all times. Otherwise, believability is lost.

In contrast, current computer games are non-linear, offering variation and interactivity. While it is possible to create characters for these games that convey a sense of psychological engagement, it is extremely difficult with existing tools.

One limitation is that there is no expert, no actor, director, animator, or puppeteer, actually present during the unfolding drama, and so authors using existing techniques are limited by what they can anticipate and produce in advance.

In this paper, we discuss the problem of building believable characters that respond to users and to each other in real time, with consistent personalities, properly changing moods, and without mechanical repetition, while always maintaining an author's goals and intentions. We describe an approach in which actors follow **scripts**, sets of author-defined rules governing their behavior, which are used to determine the appropriate animated **actions** to perform at any given time. We also describe a behavioral architecture that supports author-directed multi-actor coordination as well as run-time control of actor behavior for the creation of user-directed actors or **avatars**. Next, we describe how the system has been implemented using an English style scripting language and a network distribution model to enable creative experts, who are not primarily programmers, to create powerful interactive applications. Finally, we discuss our experiences with the system and future work.

Related Work

The phrase "Desktop Theater" was coined by Steve Strassman [Strassman91]. His philosophy was quite similar to ours. Yet because his work slightly predated the age of fast graphical workstations, it did not deal with real-time visual interaction. But there was already the emphasis on expressive authoring tools for specifying how characters would respond to direction.

The novel *Snow Crash* by Neil Stephenson also influenced this work. That novel posits a "Metaverse" a future version of the Internet which appears to its participants as a quasi physical world. Participants are represented by fully articulate human figures or avatars. Body movements of avatars are computed automatically by the system.

Snow Crash specifically touches on the importance of proper authoring tools for avatars although it does not describe those tools. Our system takes these notions further in that it supports autonomous figures that do not directly represent any participant.

Most autonomous actor simulation systems follow the parallel layered intelligence model of [Minsky86] which was partially implemented by the subsumption architecture of [Brooks86] as well as in [Bates92] and [Johnson94]. Several systems have been developed which share this layered architecture with **Improv** yet which solve distinctly different problems. The **Jack** system of [Badler93] focuses on proper task planning and biomechanical simulation as does [Hodgins95]. The general goal is to produce accurate simulations of biomechanical robots. Similarly the simulations of Terzopoulos et al [Terzopoulos94] has simulated autonomous animal behaviors that respond to their environment according to biomechanical rules. Autonomous figure animation has been studied by [Badler91] [Girard85] [Morawetz90] and [Sims94].

The **Alive** system of [Mac95] and [Blumberg95] focuses on self organizing embodied agents which are capable of making inferences and of learning from their experiences. Instead of maximizing an authors ability to express personality the **Alive** system use ethological mechanisms to maximize the actor s ability to reorganize its own personality based on its own perception and accumulated experience.

APPROACH

Improv An Expert System For Authors

As an authoring system **Improv** must provide creative experts with tools for constructing the various aspects of an interactive application. These must be intuitive to use, allow for the creation of rich, compelling content, and produce behavior at run time which is consistent with the author s vision and intentions. Animated actors must be able to respond to a wide variety of user interactions in ways that are both appropriate and non repetitive. This is complicated by the fact that in applications involving several characters, these actors must be able to work together while faithfully carrying out the author s intentions. The author needs to control the choices an actor makes and how the actors move their bodies.

ARCHITECTURE

The behavior model used by **Improv** is similar to that proposed by [Blumberg95] in that it consists of geometry that is manipulated in real time, an Animation Engine which utilizes descriptions of atomic animated actions (such as Walk or Wave) to manipulate the geometry, and a Behavior Engine which is responsible for higher level capabilities (such as going to the store or engaging another actor in a conversation) and decisions about which animations to trigger. In addition, the Behavior Engine maintains the internal model of the actor, representing various aspects of an actor s moods, goals, and personality. The Behavior Engine constitutes the *mind* of the actor. At run time, an actor s movements and behavior are computed by iterating an *update cycle* that alternates between the Animation and Behavior Engines.

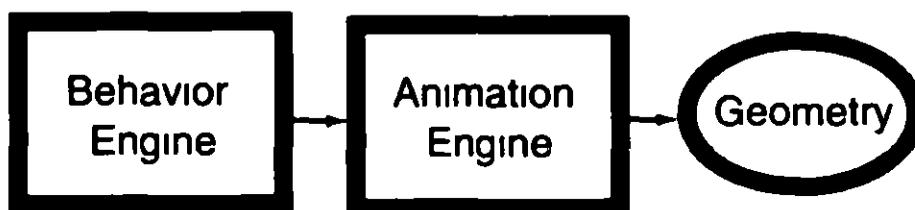


Figure 1 The basic architecture for an actor in the run time system

ANIMATION ENGINE

The Animation Engine provides tools for generating and interactively blending realistic gestures and motions. This is a generalization of the system presented in [Perlin 95]. Actors are able to move from one animated motion to another in a smooth and natural fashion in real time. Motions can be layered and blended to convey different moods and personalities. The Animation Engine controls the *body* of the actor.

Geometry

An animator can build any variety of articulated character. Actors can be given the form of humans, animals, animate objects, or fantasy creatures. An actor consists parts that are connected by rotational joints. The model can be deformable, which is useful for muscle flexing or facial expressions as illustrated in [Chadwick89].

Degrees Of Freedom

Authors specify individual actions in terms of how those actions cause changes over time to each individual *degree of freedom* (DOF) in the model. The system then combines these DOF values to make smooth transitions and layerings among actions.

There are various types of DOFs that an author can control. The simplest are the three rotational axes between any two connected parts. Examples of this are head turning and knee bending. The author can also simply position a part, such as a hand or a foot. The system automatically does the necessary inverse kinematics to preserve the kinematic chain.

From the author's point of view, the x,y,z coordinates of the part are each directly available as a DOF.

The author can also specify part mesh deformations as DOFs. To make a deformation, the author must provide a "deformation target," a version version of the model (or just some parts of the model) in which some vertices have been moved. For each deformation target, the **Improv** system detects which vertices have been moved, and builds a data structure containing the x,y,z displacement for each such vertex. For example, if the author has provided a smiling face as a deformation target, then the (s)he can declare SMILE to be a DOF. The author can then specify various values for SMILE between 0. (no smile) and 1. (full smile). The system handles the necessary interpolation between mesh vertices. In the particular case of smiling, the author can also specify negative values for SMILE, to make the face frown.



figure 2. Flexing a deformable mesh.

Continuous Signal Generation

The author defines an action simply as a list of DOFs, together with a range and a time varying expression for each DOF. Most actions are constructed by varying a few DOFs over time via combinations of sine, cosine and coherent noise. For example, sine and cosine signals are used together within actions to impart elliptical rotations.

One of the key ingredients to realism in **Improv** characters is the ability to apply coherent noise. This mechanism was originally developed for procedural textures [Perlin85][Ebert94]. In the current work it is used in essentially the same way. Using noise in limb movements allows authors to give the impression of naturalistic motions without needing to incorporate complex simulation models.

For example, coherent noise can be used to convey the small motions of a character trying to maintain balance, the controlled randomness of eye blinking, or the way a character's gaze wanders around a room. Although in real life each of these examples

has a different underlying mechanism viewers do not perceive the mechanism itself. Instead they perceive some statistics of the motion it produces. When coherent noise is applied in a way that matches those statistics, the actor's movements are believable.

The author can also import keyframed animation from commercial modeling systems such as Alias or SoftImage. The **Improv** system internally converts these into actions that specify time-varying values for various DOFs. To the rest of the system, these imported actions look identical to any other action.

Defining Actions

The author uses DOFs to build actions. Below are three different actions that define how an actor might gesture with his arm while talking. Each one uses several frequencies of noise to modulate arm movement. The first two are general hand-waving gestures, while the third shakes the arm more emphatically, as though pointing at the listener.

On each line of an action, the part name is followed first by three angular intervals, and then by three time-varying interpolants in braces. Each interpolant is used to compute a single angle in its corresponding interval. The results are applied to the part as Pitch, Roll, and Yaw rotations respectively. The angle intervals are constant over time, whereas the time-varying interpolants are reevaluated at each update cycle. For example, in the first line below, if *N0* possesses the value 0.5 at some time step, then the resulting Pitch rotation at that time step will be 0.5 of the way between 25 degrees and 55 degrees, or 40 degrees.

```
define ACTION "Talk_Gesture1"
{
    R_UP_ARM      25 55    0    -35 65    { N0 0 N0 }
    R_LO_ARM      55 95    0     0      { N1 0 0 }
    R_HAND        -40 25   75 -25   120   { N1 N2 0 }
}

define ACTION "Talk_Gesture2"
{
    R_UP_ARM      10 47    0    -10 45   { N0 0 N0 }
    R_LO_ARM      35 77    0     0      { N1 0 0 }
    R_HAND        -53 55  -40 15   120   { N1 N2 0 }
}

define ACTION "Talk_Gesture3"
{
    R_UP_ARM      45      20 15    0    { 0 N0 N0 }
    R_LO_ARM      70 120   0     0    { N1 0 0 }
    R_HAND        40 15    0     120  { N2 0 0 }
}
```

The variables *N0*, *N1*, and *N2* are shorthand that the **Improv** system provides the author to denote time-varying coherent noise signals of different frequencies. *N1* is one octave higher than *N0*, and *N2* is one octave higher than *N1*. The value of each signal varies between 0.0 and 1.0.

Note that the upper arm movement is controlled by *N0*, whereas the lower arm movement is controlled by *N1*. The result is that the upper arm will, on the average, swing back and forth about the shoulder once per second, whereas the lower arm will, on the average, swing back and forth about the elbow twice per second. Meanwhile, the hand will make small, rapid rotations about the wrist. These frequencies were chosen simply because they looked natural. In our tests, frequency ratios that varied significantly from these did not look natural. Presumably, this frequency ratio reflects the fact that the lower arm has about half as much mass as the total arm, and therefore tends to swing back and forth about twice as frequently.

Action Compositing

An **Improv** actor can be doing many things at once, and these simultaneous activities can interact in different ways. For example, an author may want an actor who is waving to momentarily scratch his head with the same hand. It would be incorrect

for the waving movement to continue during the time the actor is scratching his head. The result could be strange. For example, the actor might try to feebly to wave while his arm while making vague scratching motions about his cranium. Clearly in this case we want to decrease the amount of waving activity as we increase the scratching activity. Some sort of case in/out transition is called for.

In contrast, suppose we want an actor to scratch his head for a moment while walking downstage. It would be incorrect if the **Improv** system were to force the actor to stop walking every time he scratched his head. In this case, an case in/out transition would be inappropriate.

The difference between these two examples is that the former situation involves two actions which cannot coexist, whereas the latter situation involves two actions that can gracefully coexist. The authoring system should provide a mechanism to allow authors to make these distinctions in an easy and unambiguous way. To do this, **Improv** contains a simple set of rules. The approach we take is borrowed from image compositing methods. The **Improv** author thinks of motion as being layered, just as composited images can be layered back to front. The difference is that whereas an image maps pixels to colors, an action maps DOFs to values.

The author can place actions in different groups, and these groups are organized into a back to front order. Also, the author may select any action. Given this structure, the two compositing rules are as follows:

(1) Actions which are in the same group compete with each other. At any moment, every action possesses some weight or opacity. When an action is selected, its weight transitions smoothly from zero to one. Meanwhile, the weights of all other actions in the same group transition smoothly down to zero.

(2) Actions in groups which are further forward obscure those in groups which are further back.

Using this system, authors place actions which should compete with each other in the same group. Some actions, such as walking, are fairly global in that they involve many DOFs through the body. Others, such as head scratching, are fairly localized and involve relatively few DOFs. The author places more global actions in the rear, most groups. More localized actions are placed in front of these. Also, some actions are relatively persistent. Others are generally done fleetingly. Groups of very fleeting or temporary actions (like scratching or coughing) are placed still further in front.

For the author, this makes it easy to specify intuitively reasonable action relationships. For example, suppose the author specifies the following action grouping:

```
GROUP Stances
ACTION Stand
ACTION Walk
```

```
GROUP Gestures
ACTION No_waving
ACTION Wave_left
ACTION Wave_right
```

```
GROUP Momentary
ACTION No_scratching
ACTION Scratch_head_left
```

Then let's say actions are selected in the following order:

```
Stand
Walk
Wave_left
Scratch_head_left
No_scratching
```

Wave_right

The actor will start to walk. While continuing to walk he will wave with his left hand. Then he will scratch his head with his left hand, and resume waving again. Finally he will switch over to waving with his right hand.

Because of the grouping structure, the author has easily imparted to the actor many convenient rules. For example, the actor knows to wave with either one hand or the other (not both at once), that he doesn't need to stop walking in order to wave or to scratch his head, and that after he's done scratching he can resume whatever else he was doing with that arm.

Applying Actions To The Model

At any animation frame, the run time system must assign a unique value to each DOF for the model, then move the model into place and render it. To compute these DOFs, the algorithm proceeds as follows. Within each group, a weighted sum is taken over the contribution of each action to each DOF. The values for all DOFs in every group are then composited, proceeding from back to front. The result is a single value for each DOF, which is then used to move the model into place.

There are subtleties in this algorithm, such as correctly compositing inverse kinematic DOFs over direct rotational DOFs. But these are beyond the space limitations of this paper. For a full treatment of the DOF compositing algorithm, the reader is referred to [Perlin96].

The author is given tools to easily synchronize movements of the same DOF across actions; transitions between two actions that must have different tempos are handled by a morphing approach: During the time of the transition, speed of a master clock is continuously varied from the first tempo to the second tempo, so that the phases of the two actions are always aligned. This is similar to the approach taken by [Bruderlin95] and [Witkin95].

Action Buffering

Sometimes it would be awkward for an actor to make a direct transition between two particular actions in a group. For example, let's say the actor has his hands behind his back, and then claps his hands. Because DOFs are combined linearly, the result would be that the actor passes his hands through his body!

We allow the author to avoid such situations by declaring that some action in a group can be a buffering action for another. The system implements this by building a finite state machine that forces the actor to pass through this buffering action when entering or leaving the troublesome action.

For example, the author can declare that the action hands-at-the-sides is a buffering action for hands-behind-the-back. Then when the actor transitions between hands-behind-the back and any other action, he will always first move his hands around the sides of his body.



figure 3: Otto demonstrating action buffering.

BEHAVIOR ENGINE

Motivation

Improvisational authors cannot create deterministic scenarios because the user is a variable in the run time system. The user's responses are always implicitly presenting the actor with a choice of what to do next. Because of this variability, the user's experience of an actor's personality and mood must be conveyed largely by that actor's probability of selecting one choice over another.

As a very simple example, suppose the user often goes away for awhile, keeping an actor waiting for various amounts of time. If the actor usually sits down or naps before the user returns, then the actor will appear to the user as a lazy or tired character. The user is forming an impression based on probabilities.

The influence of the author lies in carefully tuning of such probabilities. The goal of the behavior engine is to help the author to do so in the most expressive way possible.

Mechanism

The behavior engine provides several authoring tools for guiding an actor's behavioral choices. The most basic tool is a simple parallel scripting system. Generally speaking, at any given moment an actor will be executing a number of scripts in parallel. In each of these scripts, the most common operation is to select one item from a list of items. These items are usually other scripts or actions for the actor (or for some other actor) to perform.

The real power of the behavior engine comes from probability shaping tools we provide authors for guiding an actor's choices. The more expressive the tools for shaping these probabilities, the more believable actors will be in the hands of a talented author.

In the following sections we describe the working of the behavior engine. First we describe the basic parallel scripting structure. After that we will describe the probability shaping tools.

Scripts For an Interactive World

If actions are the mechanism for continuous control of the movements made by an actor's body, then scripts are the mechanism for discrete control of the decisions made by the actor's mind.

The author must assume that the user will be making unexpected responses. For this reason, it is not sufficient to provide the author with a tool for scripting long linear sequences. Rather, the author must be able to create layers of choices, from more global and slowly changing plans to more localized and rapidly changing activities that take into account the continuously changing state of the actor's environment and the unexpected behavior of the human participant.

In the next two sections we first discuss how scripts are organized into layers, and then how an individual script operates.

Grouping Scripts

Like actions, scripts are organized into groups. However, unlike actions, when a script within a group is selected, any other script that was running in the same group immediately stops. In any group at any given moment, exactly one script is running.

Generally, the author organizes into the same group those scripts that represent alternative modes that an actor can be in at some level of abstraction. For example, the group of activities that an actor performs during his day might be

ACTIVITIES Resting Working Dining Conversing Performing

In general, the author first specifies those groups of scripts that control longer term goals and plans. These tend to change slowly over time, and their effects are generally not immediately felt by the user.

The last scripts are generally those that are most physical. They tend to choose actual body actions in response to the user and to the state of higher level scripts. For example, an actor might contain the following groups of scripts, in order, within a larger set of scripts:

DAY_PLANS Waking Morning Lunch Afternoon Dinner Evening

ACTIVITIES Resting Working Dining Conversing Performing

BEHAVIOR Sleeping Eating Talking Joking Arguing Listening Dancing

We can think of the Animation Engine with its groups of continuous actions as an extension of this grouping structure to even lower semantic levels

Individual Scripts

A script is organized as a sequence of clauses. At run time, the system runs these clauses sequentially for the selected script in each group. At any update cycle, the system may run the same clause that it ran on the previous cycle, or it may move on to the next clause. The author is provided with tools to "hold" clauses in response to events or timeouts.

The two primary functions of a script clause are 1) to trigger other actions or scripts and 2) to check, create or modify the actor's properties.

Triggering Actions and Scripts

The simplest thing an author can do within a script clause is trigger a specific action or script, which is useful when the author has a specific sequence of activities (s)he wants the actor to perform. In the following example, the actor walks onstage, turns to the camera, bows, and then walks offstage again.

```
define SCRIPT "Curtain Call"
```

```
{ "walk to center" }
{ continue until { my location equals center } }
{ "turn to camera" }
{ continue until { "turn to camera" is done } }
{ "bow" }
{ continue for 3 seconds }
{ "walk offstage" }
```

In this case, phrases in quotes represent scripts or actions. Each of these scripts might, in turn, call other scripts and/or actions. The other information (continue, etc) is used by **Improv** to control the timing of the scene.

Layered Behavior

Through layering, an author can create complex behaviors from simpler scripts and actions. Take the following example.

```
define SCRIPT "greeting"
```

```
{
{ "enter" }
{ wait 4 seconds }
{ "turn to camera" }
{ wait 1 second }
{ "wave" for 2 seconds
  "talk" for 6 seconds }
{ wait 3 seconds }
{ "sit" } { wait 5 seconds }
{ "bow" toward "Camera" }
{ wait 2 seconds }
{ "leave" }
}
```

In this example, the actor first activates the "enter" script (which instructs the actor to walk to center). The "enter" script and "greeting" script are now running in parallel. The "greeting" script waits four seconds before activating the "turn to camera" script. This tells the actor to turn to face the specified target, which in this case is the camera. The script then waits one second before instructing the actor to begin the "wave" and "talk" actions. The script waits another 3 seconds before activating the "sit" action during which time the "wave" action has ended, returning to the default "No Hand Gesture" action in its group. Meanwhile, the "talk" action continues for another three seconds after the actor sits. Two seconds later, the actor bows to the camera, waits another two seconds, and then leaves.

Non Deterministic (Stochastic) Behavior

In addition to commands that explicitly trigger specific actions and scripts **Improv** provides a number of tools for generating the more non deterministic behavior required for interactive non linear applications. An author may specify that an actor choose randomly from a set of actions or scripts as in the following example

```
SCRIPT Rock Paper Scissors
```

```
{ choose from { Rock Paper Scissors } }
```

Once an action or script is chosen it is executed as though it had been explicitly specified

Alternately the author can specify weights associated with each item in the choice. These weights are used to affect the probability of each item being chosen as in the following example

```
define SCRIPT Rock Paper Scissors2
```

```
{ choose from { Rock 5 Paper 3 Scissors 1 } }
```

In this case there is a 5/9 chance the actor executing this script will choose the Rock action 3/9 that the actor will choose Paper and a 1/9 chance the actor will pick Scissors. The decision is still random but the author has specified a distinct preference for certain behaviors over others

In order to create believable characters the author also needs to be able to have these decisions reflect an actor's mental state as well as the state of the actor's environment. An actor's decision about what to do may depend on any number of factors including mood time of day what other actors are around and what they're doing what the user is doing etc

In **Improv** authors can create **decision rules** which take information about an actor and his environment and use this to determine the actor's tendencies toward certain choices over others. The author specifies what information is relevant to the decision and how this information influences the weight associated with each choice. As this information changes the actor's tendency to make certain choices over others will change as well

Decision Rules

Properties

The information about an actor and his relationship to his environment are stored in an actor's properties. These properties may be used to describe aspects of an actor's personality such as assertiveness temperament or dexterity an actor's current mood such as happiness or alertness or his relationship to other actors or objects such as his sympathy toward the user or his attitude toward strained peas. These properties are specified by the author either when the actor is created or else within a clause of a script to reflect a change in the actor due to some action or event. The latter case is shown in the following example

```
define SCRIPT Lat Dinner
```

```
{ "Eat "  
{ set my Appetite to 0 }  
{ Belch }
```

In this case the author specifies how an actor's behavior is reflected in his personality by reducing the actor's appetite after eating

An author can also use properties to provide information about any aspect of an actor's environment including inanimate props and scenery and even the scripts and actions an actor chooses from. An author can assign properties to actions and scripts describing the various semantic information associated with them such as aggressiveness formality etc

The author can then use these values in the construction of **decision rules**. **Decision rules** allow actors to make decisions that reflect the state of the world the author has created

What Decision Rules Do

When a **decision rule** is invoked a list of objects is passed to it. The system then uses the decision rule to generate a weight between zero and one for each object. This list can then be used to generate a weighted decision.

Each **decision rule** consists of a list of author specified **factors**—pieces of information that will influence the actor's decision. Each of these **factors** is assigned a weight which the author uses to control how much influence that piece of information has upon the decision. This information can simply be the value of a property of an object as in the following example:

```
{ choose from { "Steph" "Bob" "Sarah" } based on "who's interesting" }
```

```
define DECISION RULE "who's interesting"
```

```
factor { his/her "Charisma" } influence 8
```

```
factor { his/her "Intelligence" } influence 2
```

In this example, the decision rule will use the **Charisma** and **Intelligence** properties of the three actors to generate a weight for each actor that will be used in the decision. In this case, the author has specified that the value of an actor's **Charisma** will have the greatest influence in determining that weight, with **Intelligence** having a lesser role. The influence is optional and defaults to 1.0 if unspecified. The equations for determining these weights can be found in Appendix A, *Decision Rule Equations*.

An author can also use the relationship between the actor and the various choices to influence a decision by making **"fuzzy"** comparisons between their properties. For example:

```
{ choose from ("Fight" "Flee") based on "how courageous" }
```

```
define DECISION RULE "how courageous"
```

```
{ factor { my "Courage" equals its "Courage Level" to within 5 } }
```

Here, the author is comparing the actor's **"Courage"** property with the **"Courage Level"** property associated with the scripts **"Fight"** and **"Flee"**. If the actor's **"Courage"** equals the script's **"Courage Level"**, the decision rule will assign a weight of 1 to that choice. If the values aren't equal, a weight between 0 and 1 will be assigned based on the difference between them, dropping to 0 when the difference is greater than the **"within"** range. In this case, 5. (The equations for this can be found in Appendix B, *Fuzzy Logic Equations*.) As the actor's **"Courage"** increases or decreases, so will the actor's tendency toward one option or the other.

An author may want an actor to choose from a set of options using different **factors** to judge different kinds of items. A list of objects passed to the decision rule may be divided into subsets using author-defined criteria for inclusion. The weights assigned to a given subset may be scaled, reflecting a preference for an entire group of choices over another. For example:

```
{ choose from ("Steph" "Bob" "Sarah") based on "who's interesting2" }
```

```
define DECISION RULE "who's interesting2"
```

```
{  
subset "Those I'd be attracted to" scale 1  
factor { his/her "Intelligence" equals my "Confidence" to within 4 }
```

```
subset "Those I wouldn't be attracted to" scale 8  
factor { his/her "Intelligence" equals my "Intelligence" to within 4 }  
}
```

```
define SUBSET "Those I'd be attracted to"  
{ his/her "Gender" equals my "Preferred Gender" }
```

```
define SUBSET "Those I wouldn't be attracted to"  
{ his/her "Gender" does not equal my "Preferred Gender" }
```

Let's assume the actor is considered a heterosexual male (ie his **"Gender"** is **"Male"** and his **"Preferred Gender"** is **"Female"**). The weight assigned to **"Steph"** and **"Sarah"** will depend on how closely their intelligence matches our actor's confidence (being put off by less intelligent women and intimidated by more intelligent ones, perhaps). The factor used to judge **"Bob"** reflects a sympathy toward men who are his intellectual equal, unaffected by the actor's confidence. These scale values reflect a general preference for one gender over the other.

Coordination Of Multiple Actors

Ideally we would prefer to give an author the same control over groups of actors that (s)he has over individual actors. The proper model is that the author is a director who can direct the drama via pre-written behavior rules. To the author, all of the actors constitute a coordinated "cast" which in some sense is a single actor that just happens to have multiple bodies.

For this reason, we allow actors to modify each other's properties with the same freedom with which an actor can modify his own properties. From the author's point of view, this is part of a single larger problem of authoring dramatically responsive group behavior. If one actor tells a joke, the author may want the other actors to respond favorably or not to the punchline. By having the joke teller cue the other actors to respond, proper timing is maintained, even if the individual actors make their own decisions about how exactly to react. In this way, an actor can give the impression of always knowing what other actors are doing and respond immediately and appropriately in ways that fulfill the author's goals.

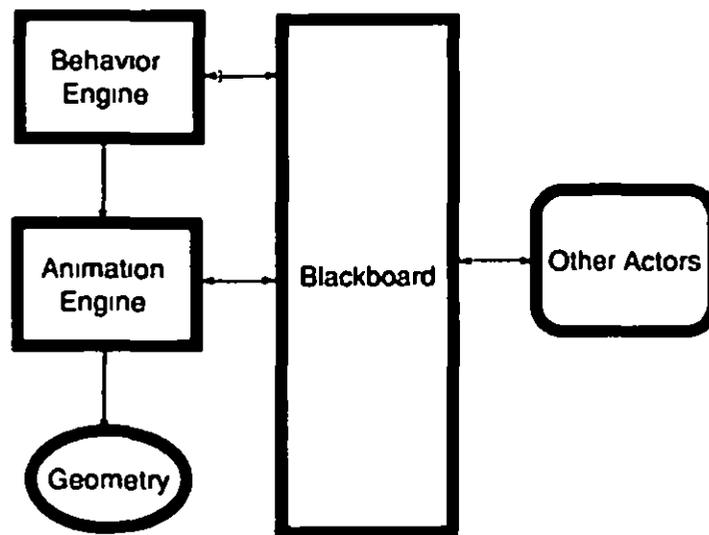


Figure 4 Actors communicate with each other through a shared blackboard

This communication occurs through the use of a shared blackboard. The blackboard allows actors to be coordinated, even when running on a single processor, on multiple processors, or across a network.

USER INTERACTION

Multi Level Control Of Actor State

Creating and Modifying User Interface Elements

An author can also include user interface specifications in an actor's scripts, enabling widgets to be easily generated at run time in response to actor behavior or to serve the needs of the current scene or interaction. The user can employ these widgets to trigger actions and scripts at any level of an actor's behavioral hierarchy. This enables users to enter the virtual environment by allowing them to direct the actions of one (or more) animated actor(s). By making this interface a scriptable element, **Improv** enables authors to more easily choreograph the interaction between the virtual actors and the human participant.

Controlling An Actor From Multiple Levels of Abstraction

One important feature of **Improv** is ability for the user to interact with the system at different semantic levels. The result of the user's actions can cause changes in the system anywhere from high level scripts to low level actions. This means that the author can give the user the right kind of control for every situation. If the user requires a very fine control over actors' motor skills, then the author can provide direct access to the action level. On the other hand, if the user is involved in a conversation, the author might let the user specify a set of gestures for the actor to use, and have the actor decide on the specific gestures from moment to moment. At an even higher level, the author may want to have the user directing large groups of actors, such as an

acting company or an army in which case (s)he might have the user give the entire group directions and leave it to the individual actors to carry out those instructions. Since any level of the actor's behavior can be made accessible to the user, the author is free to vary the level of control as necessary at any point in the application.

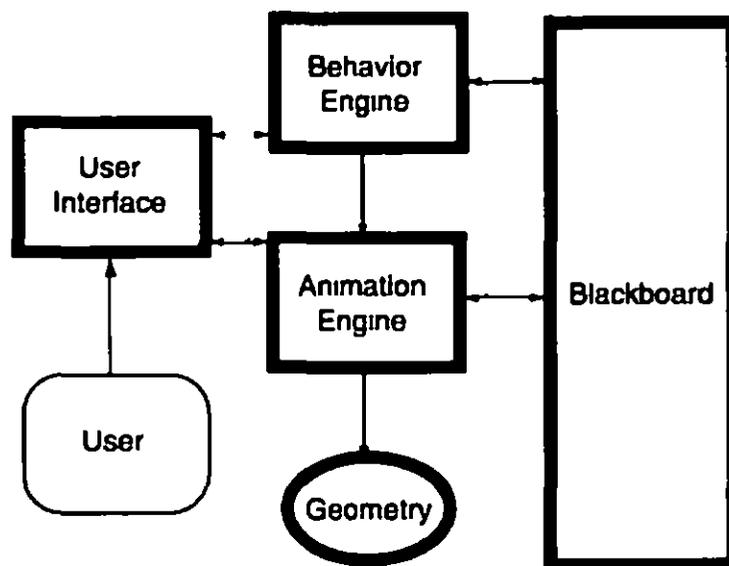


Figure 5 Users interact with both the Behavior Engine and the Animation Engine through an author defined user interface

IMPLEMENTATION

English Style Scripting Language

Many of the authors and artists interested in creating interactive content are not primarily programmers and therefore we have developed a number of "english style" scripting language extensions to *Improv* that make it easier for authors and artists to begin scripting interactive scenarios. For example, all of the code examples shown in this paper were written in the current *Improv* syntax.

Because the scripting language is written as an extension of the system language, as users become more experienced they can easily migrate from scripting entirely using the high level english style syntax to extending the system through low level algorithmic control.

Network Distribution

Improv is implemented as a set of distributed programs in UNIX, connected by TCP/IP socket connections, multicast protocols, and UNIX pipes. The participating processes can be running on any UNIX machines. This transport layer is hidden from the author.

All communication between participant processes is done by continually sending and receiving programs around the network. These are immediately parsed into byte code and executed. At the top of the communication structure are routing processes. There must be at least one routing process on every participating Local Area Network. The router relays information among actors and renderer processes. For Wide Area Network communication, the router opens sockets to routers at other LANs.

In our current implementation, each actor maintains a complete copy of the blackboard information for all actors. If an actor's behavior state changes between the beginning and end of a time step, then these changes are routed to all other actors.

Virtual Simultaneity

Typical Wide Area Network (WAN) latencies can be several seconds. This poses a problem for two virtual actors interacting in a distributed system. From the viewpoint of believability, some latency is acceptable for high level decisions but not for low level physical actions. For example, when one character waves at another, the second character can get away with pausing for a moment before responding. But two characters who are shaking hands cannot allow their respective hands to move through

space independently of each other. The hands must be synchronized to at least the animation frame rate.

The blackboard model allows us to deal with this situation gracefully. We can split the Behavior Engine and Animation Engine for an actor across a Wide Area Network, and have these communicate with each other through the blackboard. For the DOFs produced by the Animation Engine, we allow the blackboard to contain different values at each LAN. For the states produced by the Behavioral Engine, the actor maintains a single global blackboard.

Computationally, each actor runs the Behavioral Engine at only a single Local Area Network (LAN) but duplicates Animation Engine calculations at each LAN. When two characters must physically coordinate with each other, then they use the local versions of their DOFs. In this way, an actor is always in a single Behavioral State everywhere on the WAN, even though at each LAN he might appear to be in a slightly different position. In a sense, the actor has one mind but multiple bodies, each inhabiting a parallel universe. Although these bodies may differ slightly in their position within their own universe, they are all consistent with this one mind.

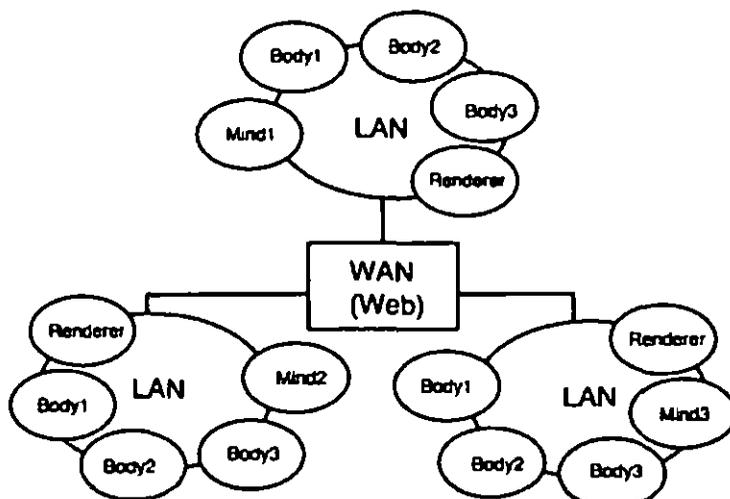


Figure 6 Wide Area Network Distribution Model

This leads to an interesting and fundamental property. Let us suppose that our **Improv** actor Gregor is dancing while balancing a tray in an **Improv** scene. I further suppose that the scene is being watched at the same time by people in Sao Paulo, Brazil, and in Manhattan, New York. Perhaps some of these people are interacting with Gregor. The connection is through the Internet.

In this scene, Gregor's Behavior Engine makes all the choices about whether to dance, whether to keep balancing the tray, how much joy and abandon versus self-conscious restraint he puts into the dance. His Animation Engine must set all the DOFs that determine how he moves when doing these things, so as to be responsive and coordinated.

If the people in NY and those in SP are talking on the telephone, they will report seeing the same thing. Yet, if a high-speed dedicated video link were established and participants could see the two Gregors side by side, they would see two somewhat different animations. In one, Gregor's hand might thrust up to balance the tray half a second sooner; in the other, he might have his other arm extended a bit further out. He might be rocking right to left on one screen, while he is rocking from left to right on the other.

Thus, everywhere in the world, there is only one social Gregor. He has a single mood, a single personality; he is only engaged in one task. Yet Gregor can have many slightly different physical realities, differing only up to the threshold where they might disrupt the social unity of his Behavioral State.

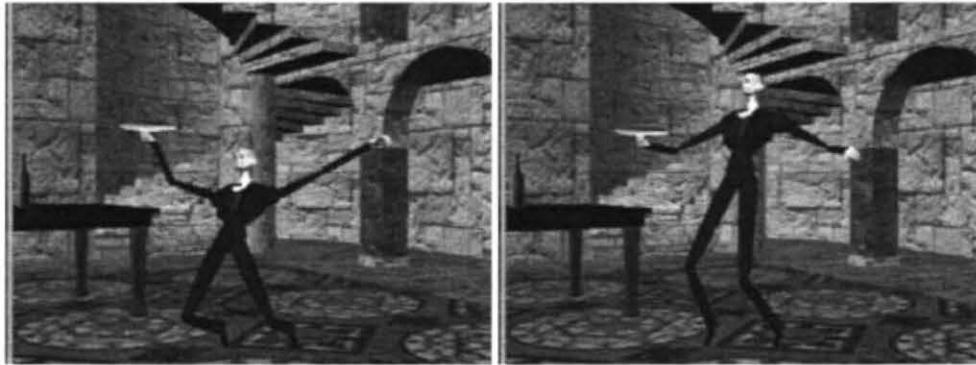


figure 7: Two versions of Gregor dancing, each on different networked computer.

In fact, if communication lag exceeds several seconds, significant differences may have occurred between the various Gregor instances. This can lead to problems. For example, suppose two actors that are temporarily out of communication each try to pick up some physical object.

This is a standard collaborative work dilemma. The only reliable solution is to make the object itself an actor (albeit a light weight one). As odd as it seems, the object itself must agree to be picked up, since it too must maintain a consistent physical reality. This was also independently observed by [Karaul95].

Communicating with Improv Actors From Outside The System

The blackboard protocol has a great advantage in terms of flexibility. To take full advantage of this flexibility, we provide a C support library that gives access to the blackboard. This allows researchers who know nothing about the **Improv** system, except for the names of actions and scripts, to begin immediately to control **Improv** actors.

For example, a researcher can write a standalone C program that links with the support library. The program can pass string arguments such as "Gregor Sit" or "Otto Walk_To_Door" to an output function. This is all that the program needs to do, in order to modify actors' behavior states.

Since the system treats the standalone program as just another actor, the program can also listen for messages by calling an input routine. These messages contain the information that updates the blackboard, saying where various actors are, what they are doing, what their moods are, etc.

In practice, this allows researchers and students at other institutions who know nothing about **Improv** except its GUI to immediately begin to use the system for their own applications. In our research collaborations we find that this is a highly successful way for our collaborators to bootstrap.

Improv also has several audio subsystems. These subsystems are used for speech generation, music generation, allowing actors to follow musical cues, and generating ambient background noise.

Extended Example

The following is an example of a scene involving multiple actors involved in a social interaction with a user.

```
define SCRIPT "Tell Joke"
{
  { do "Turn to Face" to
choose from { others except player }
}
  { cue { others except player } to "Listen To Joke" to me }
  {
do "No Soap, Radio"
do "Joke Gestures" }
  { wait until { current "Joke" is "completed" } }
  { do "Laugh" for 3 seconds }
```

```

{ cue { others except player } to React To Joke }
{ wait 3 seconds }
{ do React To Player }
}

```

In this example the actor executing the script randomly chooses one of the actors not being controlled by the user and turns to him or her. The actor then cues the other non user actors to execute the Listen To Joke script in which the actor chooses the appropriate gestures and body language that will give the appearance of listening attentively.

```

define SCRIPT Listen To Joke
{
{
choose from { entire set of Stances } based on appropriate listening gestures
choose from { entire set of Gestures } based on appropriate listening gestures
}
{ continue for between 3 and 12 seconds }
{ repeat }
}

```

Here the actor chooses from the actions in the of "Stances and Gestures" using the decision rule appropriate listening gestures.

```

define DECISION_RULE appropriate listening gestures
{
subset Listening? scale 1
factor { my confidence is greater than its confidence to within 0.3 } influence 5
factor { my self control is less than its self control to within 0.3 } influence 5 }

```

```

define SUBSET Listening?
{ it is reactive and conversational or generic }

```

In this rule the actor narrows the list down to those actions that are reactive and conversational or generic actions that can be used in any context. The rule then compares the confidence and self control of the actor those assigned to each action creating a weighted list favoring actions that match the fuzzy criteria. After choosing from the list the actor will wait from 3 to 12 seconds before repeating the script and choosing another gesture.

Meanwhile The actor telling the joke then executes the No Soap Radio script which contains a command to an external speech system to generate the text of the joke. At the same time the actor executes the Joke Gestures script which like the Listen To Joke script chooses appropriate gestures based on the actor's personality.

The actor continues until the joke is finished (the speech system sends a command to set the script's completed property to true) and then laughs cueing the other actors to execute the "React To Joke" script.

```

define SCRIPT React To Joke
{
{ choose from { Laugh Giggle Ignore Get Upset } based on feelings toward player }
}

```

```

define DECISION_RULE feelings toward player
{ factor { my sympathy toward player does not equal its mood to within 4 } }

```

Simply put the more sympathy actor have for the player the less likely they are to react positively to the joke.

Finally the actor executes the "React To Player" script in which the actor chooses an appropriate reaction to the player depending on whether or not the player tells his actor to laugh. If he does the joke teller laughs maliciously if her sympathy for the player is low playfully if her sympathy for the player is high. If the player's actor doesn't laugh the joke teller executes the "Get It" script taunting the player until he gets mad and/or leaves.

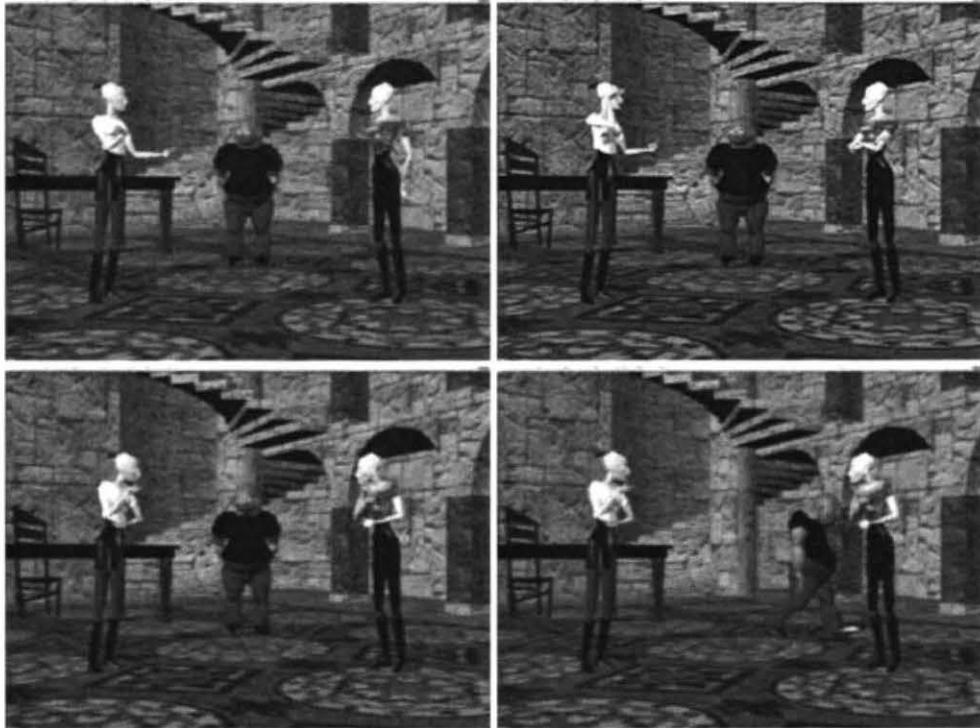


figure 8: Izzy tells Otto (the user) and Elli a joke. Elli is amused, Otto isn't.

EXPERIENCE

SIGGRAPH 95

At SIGGRAPH 95 we demonstrated an interactive embodied actor named Sam who responded to spoken statements and requests. Voice recognition was provided by *DialectTech*, a company that has developed an interface for an IBM continuous speech recognition program. In our demonstration, untrained participants could conduct a game of "Simon Says". Sam would follow requests only if they were preceded by the words "Simon Says". To make it more interesting we programmed Sam so that sometimes he would also follow requests not preceded by "Simon Says", but then he would act embarrassed at having been fooled. Our experience was that the sense of psychological involvement by participants was very great and compelling. Participants appeared to completely "buy into" Sam's presence. We believe that this was due to several factors:

- (i) participants could talk with Sam directly,
- (ii) participants knew Sam was not being puppeteered (the participant was the only human in the interaction loop), and
- (iii) Sam's motions were relatively lifelike and never repeated themselves precisely.

We have also found that allowing the participant to appear as an embodied avatar enhances the participant's sense of fun and play, and therefore involvement. We had positive experience of this at SIGGRAPH 95. We presented the participant with a large rear projection of a room full of embodied conversational agents. The participant's position, as well as simple arm gestures, were tracked by an overhead video camera. The participant appeared in the scene as a flying bat. As the participant walked around, the bat flew around accordingly. The nearest agent would break out of conversing with the other agents, and begin to play with the bat. When the participant flapped his/her arms, the bat would fly higher in the scene, and the camera would follow, which gave the participant a sense of soaring high in the air. We found that participants, and children in particular, enjoyed this experience very much, and would spend long periods of time "being" the bat and flying in turn around the heads of each of the embodied agents.



figure 9: Participant interacting with **Improv** actors as a bat.
From *SIGGRAPH 95* Interactive Entertainments Exhibit.

Other Users

We have also provided a copy of **Improv** to a number of researchers at other Universities. These researchers are pursuing their own research on top of our actor embodiment substrate. In at least one case, they plan to do comparisons with their own existing agent embodiment system.

Feedback from these collaborators on the use of **Improv** indicates that it is a useful tool for the embodiment of intelligent actors, especially for study of social interaction. In particular, it was suggested as a good tool for building educational VR environments, when used in conjunction with research software for virtual Interactive Theater. The combination can be used to simulate behaviors that would be likely to engage children to respond to, identify with and learn from knowledge agents.

We have added extensions to **Improv** so that animators can use commercial tools, such as Alias and SoftImage, to create small atomic animation components. Trained animators can use these tools to build up content. Such content can include various walk cycles, sitting postures, head scratching, etc. The procedural animation subsystem is designed in such a way that such action styles can be blended. For example, two or three different styles of walks can be separately designed from commercial key frame animation packages, and then blended together, or else blended with various procedural walks, to create continuously variable walk styles that reflect the actor's current mood and attitude, as well as the animator's style.

FUTURE DIRECTIONS

It is well known in traditional animation that human motions are created from combinations of temporarily overlapping gestures and stances. One of our current goals is to use **Improv**'s ability to tie into commercial animation tools to build up a library of component motions, and to classify these motions in a way that makes them most useful as building blocks.

We have begun to embed **Improv** into a client-based application for a Java compatible browser (such as Netscape version 2.0). For use in educational and home settings, we plan to augment the full 3D subsystem with a "nearly 3D" version. This would run on a low end platform, such as a PC with an Intel processor. The user would still be able to see a view into a three dimensional world, but the visual representations of the actors would be simpler and largely two dimensional. For example, two participants to a graphical MUD, one with an SGI Onyx, and one with an Intel/486 based PC, could interact in the same scene. They would both see the same actors at the same locations, actions and personality. The only difference would be that the first participant would see a much more realistic quality of rendering.

We plan to integrate **Improv**'s voice recognition and english-like behavioral sub-systems. This will allow a user to fully exploit the object substrate, giving access to direction of goals, mood changes, attitudes and relationships between actors, all via spoken

English sentences

CONCLUSION

We have described an interactive system that lets authors of various abilities create remarkably lifelike responsively animated character interactions that run over networks in real time. We believe these techniques have the potential to have a large impact on many areas. These include computer Role Playing Games, simulated conferences, "clip animation," graphical front ends for MUDs, synthetic performance, shared virtual worlds, interactive fiction, high level direction for animation, digital puppetry, computer guides and companions, point to point communication interfaces, true non linear narrative TV, and large scale deployment of bots for the Metaverse.

As *Improv* is a very large system, we could not cover many of its details in this paper. We refer the reader to [Perl96] for a more in depth treatment.

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of Microsoft Corporation (and especially Dan Ling), the National Science Foundation, the New York State Science and Technology Foundation, and Silicon Graphics Incorporated (especially Keith Seto). Daniel Wey and Jon Meyer have both made important contributions to the *Improv* substrate. Mauricio Oka designed the flexible face model. Many other people have helped with this effort in many ways. In particular, we'd like to thank Cynthia Allen, Lilly Castiglia, Troy Downing, Steve Feiner, Laura Haraly, Mehmet Karaul, Sabrina Liao, Marcelo Tocci, More Ruggero Ruschioni, Eduardo Toledo Santos, Jack Schwartz, Gerry Seidman, Eric Singer, Michael Wahrman, and Marcelo Zuffo. Also everyone at the CAT and MRL at NYU, and LSI at USP, E Emi, com.br/ijos.

APPENDICES

A Decision Rules Equation

When an object is passed through a decision rule, a weighted sum is made of each of the values returned from the associated factors, modified by the scale assigned to the set of choices. This becomes the final weight assigned to the object that is used in making the decision.

The formula for this is as follows:

$$\text{FinalWeight} = \text{Scale}(\text{factor1influence1} + \text{factor2influence2} + \dots + \text{factorninfluence n})$$

B Fuzzy Logic Equations

The function compares how close the *Input Value* comes to the *Target Value* (or *Target Range*), returning a value of 1 at the *Target Value* (or inside the *Target Range*), dropping to 0 at a distance of *Spread* from the *Target Value*. The fuzzy comparison is implemented as follows:

$$y = w \left(\frac{|\text{Input Value} - \text{Target Value}|}{\text{Spread}} \right)$$

where

y is the Fuzzy Value

w is a bell curve weighting kernel (we use a raised cos function)

A high and low spread may be specified, in which case input values greater than the target value (or range) will use the high spread in the calculation, while input values lower than the target value (or range) will apply the low spread.

The returned value is then modified based on the type of fuzzy operation as follows:

equals	y Value
not equals	1-y, its complement
greater than	y high spread defaults to infinity
not greater than	1-y high spread defaults to infinity
less than	y, low spread defaults to infinity
not less than	1-y low spread defaults to -infinity

REFERENCES

- N Badler B Barsky D Zeltzer *Making Them Move Mechanics Control and Animation of Articulated Figures* Morgan Kaufmann Publishers San Mateo CA 1991
- N Badler C Phillips B Webber *Simulating Humans Computer Graphics Animation and Control* Oxford University Press 1993
- J Bates A Loyall W Reilly *Integrating Reactivity Goals and Emotions in a Broad Agent* Proceedings of the 14th Annual Conference of the Cognitive Science Society Indiana July 1992
- B Blumberg T Galyean *Multi Level Direction of Autonomous Creatures for Real Time Virtual Environments* Computer Graphics (SIGGRAPH 95 Proceedings) 30(3) 47 54 1995
- A Broderlin L Williams *Motion Signal Processing* Computer Graphics (SIGGRAPH 95 Proceedings) 30(3) 97 104 1995
- R Brooks *A Robust Layered Control for a Mobile Robot* IEEE Journal of Robotics and Automation 2(1) 14 23 1986
- J Chadwick D Haumann R Parent *Layered construction for deformable animated characters* Computer Graphics (SIGGRAPH 89 Proceedings) 23(3) 243 252 1989
- D Ebert and et al *Texturing and Modeling A Procedural Approach* Academic Press London 1994
- M Girard A Maciejewski *Computational modeling for the computer animation of legged figures* Computer Graphics (SIGGRAPH 85 Proceedings) 20(3) 263 270 1985
- J Hodgins W Wooten D Brogan J O'Brien *Animating Human Athletics* Computer Graphics (SIGGRAPH 95 Proceedings) 30(3) 71 78 1995
- M Johnson *WavesWorld PhD Thesis A Testbed for Three Dimensional Semi Autonomous Animated Characters* MIT 1994
- M Karaul *personal communication*
- P Maes T Darrell and B Blumberg *The Alive System Full Body Interaction with Autonomous Agents* in Computer Animation 95 Conference Switzerland April 1995 IEEE Press pages 11 18
- M Minsky *Society of Mind* MIT press 1986
- C Morawetz T Calvert *Goal directed human animation of multiple movements* Proc Graphics Interface} pages 60 67 1990
- K Perlin *An image synthesizer* Computer Graphics (SIGGRAPH 85 Proceedings) 19(3) 287 293 1985
- K Perlin *Danse interactif* SIGGRAPH 94 Electronic Theatre Orlando
- K Perlin *Real Time Responsive Animation with Personality* IEEE Transactions on Visualization and Computer Graphics 1(1) 1995
- K Perlin A Goldberg *The Improv System* Technical Report NYU Department of Computer Science 1996 (online at <http://www.mrl.nyu.edu/improv>)

- K Sims *Evolving virtual creatures* Computer Graphics (SIGGRAPH 94 Proceedings) 28(3) 15-22 1994
- N Stephenson *Snow Crash* Bantam Doubleday New York 1992
- S Strassman *Desktop Theater Automatic Generation of Expressive Animation* PhD thesis MIT Media Lab June 1991
(online at http://www.method.com/straz/straz_phd.pdf)
- D Terzopoulos, X Tu, and R Grzeszczuk *Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World* Artificial Life 1(4) 327-351 1994
- A Witkin, Z Popovic *Motion Warping* Computer Graphics (SIGGRAPH 95 Proceedings) 30(3) 105-108 1995

Real time responsive animation with personality

Ken Perlm
Media Research Laboratory
Department of Computer Science
New York University
715 Broadway NY NY 10003
perlm@nyu.edu

Abstract

Building on principles from our prior work on procedural texture synthesis, we are able to create remarkably life-like, responsively animated characters in real time. Rhythmic and stochastic noise functions are used to define time-varying parameters that drive computer-generated puppets. Because we are conveying just the “texture” of motion, we are able to avoid computation of dynamics and constraint solvers.

The subjective impression of dynamics and other subtle influences on motion can be conveyed with great visual realism by properly-tuned expressions containing pseudo-random noise functions. For example, we can make a character appear to be dynamically balancing herself, to appear nervous, or to be gesturing in a particular way.

Each move has an internal rhythm, and transitions between moves are temporally constrained so that “impossible” transitions are precluded. For example, if while the character is walking we specify a dance turn, the character will always step into the turn onto the correct weight-bearing foot. An operator can make a character perform a properly-connected sequence of actions, while conveying particular moods and attitudes, merely by pushing buttons at a high level.

Potential uses of such high-level, textural approaches to computer graphic simulation include Role-Playing Games, simulated conferences, clip animation, graphical front ends for MUDs (Ste92) (Ger92), and synthetic performances.

1 Introduction

1.1 Description of the Problem

In previous work (Per85) we used pseudo-random functions to create natural surface textures of surprisingly realistic appearance without having to model the underlying physics. This was done with a set of interactive tools, consisting of

- a powerful interactive prototyping language
- a good set of signal generation and modification functions
- a good controllable noise primitive
- a set of conventions for fitting things together

In the recent work described in this paper we have applied this approach to the problem of building real time graphic puppets that appear to be emotionally responsive.

We choose the word "puppets" very deliberately here. This work is not artificial intelligence - these animated characters do not encode any real intentionality - they only encode a visual impression of personality. This work was first presented in (Pcr91). We will refer to the "dancer" figure from that animation in our examples.

1.2 Related work

Simulated actors that embody true physical constraints are being developed by Badler et al at the University of Pennsylvania (BPW93). Dynamic balancing walking robots have been developed by Rubert (Re786) and animal and human figure simulations based on inverse dynamics have been developed by Girard (GM85).

Using layered construction in the design of articulated movement has been explored by Chadwick et al (CHP89). Similarly layered control structures for walking robots (subsumption architectures) have been used effectively by Brooks (Bro86).

Morawetz and Culvert have added a sense of personality to simulated human movements by supporting secondary movements (MC90). In a radically different approach genetic algorithms have been seen to induce movement that gives an intriguing impression of personality in goal directed mutations of articulated figures (Sim91).

1.3 Guiding principles

We adopt the following general approach. Program individual motions into the puppets beforehand, but also ensure that transitions between any pair of actions are visually correct.

A potential objection to this approach is that things might look repetitious. But by using randomization we can easily build actions that are very controllable yet never actually repeat themselves.

In addition to the forward kinematics of individual actions we build in only three simple constraints. Characters never walk through walls, they never spin their heads all the way around backwards, and they maintain fixed foot contact with the floor when doing "walking" actions.

1.4 Comparison with dynamics approaches

This approach has advantages as well as disadvantages when compared with more ambitious approaches that try to model the underlying physics. One advantage is that it allows much more direct control over the subtle movements that convey the appearance of emotional expressiveness. Another is that computational costs are far lower.

A disadvantage is that the model can't teach itself new actions. If the puppet's foot is snagged by a rock while walking, the puppet cannot properly perform the particular

movement of tripping and recovering its balance unless we've already taught it how to do that.

The two approaches are compatible. Ideally, a system would employ both predetermined movements as well as physical laws that allow it to deal with unexpected events in its environment, such as the sorts of dynamic balancing of the Tuck figure from Badler's group at U. Penn.

2 Method

2.1 Actions, Weights, Transitions

The two major components of our method are *actions* and *weights*. An *action* is some simple or repetitive movement, such as walking or standing, or a pirouette turn. The relative contribution of an action is given by a *weight*, which is always a scalar value between 0.0 and 1.0.

We cause the puppet to respond to her environment in real time mainly by changing these various weights. For example, decreasing the weight for one action while simultaneously increasing the weight for another causes a transition in behavior from the first action to the second [figure 1].

Note that if this is done properly, the actions can be connected together seamlessly in arbitrary sequences, like characters in a string of text, to build up complex behaviors [figure 2]. In spirit this is similar to the summation of B-spline knot functions to construct smooth piecewise cubic curves.

If these transitions are applied naively, the results can be disastrous. An example would be a transition from an action in which our puppet is stepping onto her left foot to an action in which she is stepping onto her right foot. Another example would be a transition that would make the puppet's arm interpenetrate her body on its way from the first action to the second action.

We solve this problem by controlling the times of transitions between actions, and by designing actions in such a way that it is possible to control their transitions.

In this section we will describe the structure of our system, proceeding in a bottom-up fashion. We start with joint kinematics, go on to explain how coherent actions are developed, and then to the scalar controls that combine those actions. The bottom-to-top structure of our system is as follows:

- joint kinematics

- individual actions

- scalar weights to blend actions

- synchronizing actions

- discrete choice controls

layers of choice controls

constraints

2.2 Bottom level kinematic hierarchy

19 universal joints are used in our current approximation of the human figure—one each for wrist, neck, and head, plus four for each limb (figure 3). Ideally there should be more; this was the minimum that seemed necessary to allow emotional expressiveness.

Separate joints appear at the base and top of the neck. The first universal “arm” joint is actually at the chest. This controls the position of the shoulder. The arm structure involves the following universal joints: chest, shoulder, elbow, and wrist. Similarly, a leg has pelvis, hip, knee, and ankle joints.

Each universal joint allows three rotations: x , then z (the two “aiming” parameters) followed by y (rotation around the limb axis). Any angle not specified defaults to a “zero” position where the puppet is standing upright with arms at her side.

Here is the actual code of the main routine for positioning and drawing the body, executed once per frame, as expressed in our modeling language’s reverse polish notation.

Push

Neck joint

Nod joint draw_head

Push

Lchest joint

Lshoulder joint

Lelbow joint

Lwrist joint draw_arm

-1 1 1 scale

Rchest joint

Rshoulder joint

Relbow joint

Rwrist joint draw_arm

Pop

Waist draw_torso

Push

Lpelvis joint

Lhip joint

Lknee joint

Lankle joint 1 draw_leg

-1 1 1 scale

Rpelvis joint

```

Rhip   joint
Rknee  joint
Rankle joint  2 draw_leg

```

```
Pop
```

```
Pop
```

Push and Pop manipulate a local matrix stack as in the Silicon Graphics GI model (SC191). Wust, Neck, Nod, Tchest, etc. are joint variables, and draw_head, draw_arm, draw_torso, draw_leg, are procedures to draw parts of the body. Note the scaling by -1 in the x dimension to draw the right arm and leg as mirrors of the left arm and leg.

The variables Wust, Neck, Nod, etc. represent the 19 universal joints of the figure. Each is a vector of length three. The values in these vectors, which change at every frame, drive the figure's joints. They are set by actions and by transitions between actions.

The actual work is done in subprocedures draw_head, draw_torso, draw_arm, and draw_leg. Each of these routines does successive forward kinematic transformations on the current matrix in order to compute locations for the puppet's component parts.

2.3 Actions

A primitive action is constructed by varying the puppet's scalar joint angles over time t via expressions of raised sine and cosine, as well as noise, where raised sine and raised cosine are defined by $\frac{1+\sin\theta}{2}$ and $\frac{1+\cos\theta}{2}$.

At each frame we compute raised sine and cosine as a function of time at two frequencies one octave apart:

```

s1 = rsin(time)
c1 = rcos(time)
s2 = rsin(2*time)
c2 = rcos(2*time)

```

The variables $s1$ and $c1$ are used together within actions to impart elliptical rotations. Variables $s2$ and $c2$ do the same at double frequency.

Together the expressions $s1$, $c1$, $(1 - s1)$, and $(1 - c1)$ collectively generate a four phase periodic signal. In practice we have not found any need for finer phase control of periodic actions than this quarter cycle accuracy.

We also provide a set of independent coherent noise sources $n1$, $n2$:

```

n1 = 5 * (1 + noise(t))
n2 = 5 * (1 + noise(t + 100))
n3 = 5 * (1 + noise(t + 200))

```

The coherent noise source is defined as in (Per8). Here the noise is simpler since it need only be defined over a one dimensional temporal domain rather than over a three dimensional spatial domain. The algorithm we use is

- (1) if x is an integer $noise(x) = 0$
- (2) define a mapping $G(t)$ from the integers to a fixed set of pseudorandom gradients
- (3) given any $t < x < t + 1$ do a hermite spline interpolation using the two neighboring gradients $G(t)$ and $G(t + 1)$

The only tricky step above is (2). To implement this step efficiently we precompute a table of pseudorandom gradients $g[0..255]$. Then for any integer t we return $g[t_{mod}256]$. A comprehensive discussion of noise implementations can be found in (Fe191).

Every action is built by using some combination of the above source signals in simple expressions to control the rotation of some joints within some range. Each joint is at its zero position when the puppet is standing at attention with both arms at the side. An action is specified by a table of ranges and time dependent behavior for each joint that this action affects.

The stylized way in which these are coded makes it simpler to provide high level descriptions of rhythmic motions. Here is the code we use to specify a "rhumba" dance (figure 1).

```
{
  { 5 5 5 } { -5 -5 -5 } { n1 n2 n3 } Nod
  { 15 0 5 } { -15 0 -5 } { c1 0 s1 } Rchest
  { 0 0 0 } { 0 0 0 } { s1 0 s1 } Rshoulder
  { -90 0 0 } { -70 0 0 } { s1 0 s1 } Relbow
  { 0 0 0 } { 0 0 0 } { s1 0 s1 } Rpelvis
  { -25 -15 5 } { 0 0 -10 } { s1 s1 s1 } Rhip
  { 50 0 0 } { 0 0 0 } { s1 0 s1 } Rknee
  { 0 0 0 } { 0 0 0 } { s1 0 s1 } Rankle
  { 0 0 10 } { 0 0 -10 } { s1 0 s1 } Waist
  { -15 0 -5 } { 15 0 5 } { c1 0 s1 } Lchest
  { 0 0 0 } { 0 0 0 } { s1 0 s1 } Lshoulder
  { -70 0 0 } { -90 0 0 } { s1 0 s1 } Lelbow
  { 0 0 0 } { 0 0 0 } { s1 0 s1 } Lpelvis
  { 0 0 -20 } { -10 -25 20 } { s1 s1 s1 } Lhip
  { 0 0 0 } { 20 0 0 } { s1 0 s1 } Lknee
  { 0 0 0 } { 0 0 0 } { s1 0 s1 } Lankle
} 'rhumba define_action
```

Each line of the above code specifies an assignment of three items to a particular joint (Nod, Rchest, etc). Each of these items contains three numeric values. Each of the first two items is run immediately and packaged up as a vector representing an extreme position of motion for the joint. The third item is evaluated at every frame where the action is performed and is used as a linear interpolant between the two extremes.

Let us take the second line as an example. It specifies motion for the "Rchest" joint, the universal joint which pivots around the chest to displace the right shoulder. For this joint

the limits of rotation about the x axis are -15 degrees to 15 degrees. Similarly, the y axis is fixed at 0 degrees, and the z axis varies from 0 degrees to 90 degrees.

The time-varying behavior for this joint is as follows. The x axis interpolates between its limits as $cl = r \cos(time)$ and the z axis interpolates between its limits as $sl = r \sin(time)$. The y axis stays fixed.

The action defined above is a relatively stylized dance step, so most of its motion is rhythmic, controlled by periodic functions. Only the head motion has a little randomness, which in this case gives the impression that the puppet is looking around while she dances.

Notice also that the joints at the chest are driven by cl in their x axis and by sl in their z axis. This gives an elliptical motion to the shoulders, which is crucial for giving the subtly Latin feel of this dance move.

In contrast, here is the definition for standing in a casual pose [figure 5].

```
{
  { 0 15 0 } { 0 -15 0 } { 0 n1 0 } Neck
  { 20 0 0 } {          } {          } Nod
  { 0 0 -5 } {          } {          } Lchest
  { 0 0 0 } {          } {          } Rchest
  { -10 0 0 } {          } {          } Lshoulder
  { -10 0 0 } {          } {          } Rshoulder
  { 0 0 -10 } {          } {          } Lelbow
  { 0 0 -10 } { 0 0 -5 } { 0 0 n1 } Relbow
  { 0 0 5 } {          } {          } Waist
  { -2 0 2 } { 2 0 -2 } { n1 0 n1 } Lpelvis
  { -2 0 -2 } { 2 0 2 } { n1 0 n1 } Rpelvis
  { 0 0 -14 } {          } {          } Lhip
  { -10 25 12 } {          } {          } Rhip
  { -5 0 0 } {          } {          } Lknee
  { 25 0 0 } {          } {          } Rknee
} 'stand define_action
```

Here most of the vectors are left blank. This means that these joints are completely static for this action. All of the motion of this action is driven by noise—there is no rhythmic motion at all. The noise gives the effect of subtle restlessness and weight shifting. The motion is subtle, but if it is left out, the puppet looks stiff and unrealistic.

For some actions, we put small additional expressions in the third, time-dependent, vector in order to couple actions between the joints, or to modify the bias or gain of a joint (PerS5). For example, here is the specification of a running action. For clarity of exposition, we have assigned the four phase signals to variables A , B , C , and D , respectively. We have also assigned double-speed oscillation and its complement to variables $A2$ and $B2$, respectively.

```
{
```

```

c1      => A
s1      => B
1 A -   => C
1 B -   => D
c2      => A2
1 A2 -  => B2

```

```

{  0 -15  0 } {  5 15 0 } { A2      C 0 } Waist
{  0 -90  0 } {  0 90 0 } {  0      N 0 } Head
{  0 -10 -5 } {  0 10 5 } {  0      C D } Rchest
{  0  0  0 } { 45  0 0 } { D      0 0 } Rshoulder
{ -120  0 -10 } {  0  0 0 } { C      0 B2 } Relbow
{ -10  0  0 } { 10  0 0 } { C      0 0 } Rwrist
{  0 -10 -5 } {  0 10 5 } {  0      A B } Lchest
{  0  0  0 } { 45  0 0 } { B      0 0 } Lshoulder
{ -120  0 -10 } {  0  0 0 } { A      0 B2 } Lelbow
{ -10  0  0 } { 10  0 0 } { A      0 0 } Lwrist
{  0 -10  0 } {  0 10 0 } {  0      A 0 } Rpelvis
{ -40  0  0 } { 40  0 0 } { A B  3 * - 0 0 } Rhip
{  0  0  0 } { 130  0 0 } { B  2 bias 0 0 } Rknee
{ -45  0  0 } { 45  0 0 } { C  7 bias 0 0 } Rankle
{  0 -10  0 } {  0 10 0 } {  0      C 0 } Lpelvis
{ -40  0  0 } { 40  0 0 } { C D  3 * - 0 0 } Lhip
{  0  0  0 } { 130  0 0 } { D  2 bias 0 0 } Lknee
{ -45  0  0 } { 45  0 0 } { A  7 bias 0 0 } Lankle

```

```

} 'running define_action

```

This action consists entirely of rhythmic motion except for the head's "looking around" movements which are driven by coherent noise. The double speed oscillations serve to slightly bend and unbend the waist twice per cycle as weight is alternately borne either by one foot or by two feet. Double speed oscillations are also used to rotate slightly about the elbow's y axis. This rotation pulls the forearms a bit closer to the body twice per cycle both when in front and when behind the body.

Note the use of the bias function in the rotations about the knee joints. These give the non weight bearing knee a little extra kick at the time it swings forward. For the same reason we add a small amount of rotation to each hip 90 degrees out of phase with its primary motion.

Once an action is designed this sort of structure provides many opportunities for customization. In the above example we can replace each of the constants in the knee and hip bias expressions by variables. As we modify these variables we obtain walks that reflect different emotive states and degrees of energy.

2.4 Combining multiple weighted actions

To combine actions we assign numerical weights to every potential action in the action mix. These weights give the relative contribution of each action to the total motion of the puppet. The weights vary over time – at any given moment only a few actions have a non-zero weight. For each joint, the contributions from all the actions to that joint's position are combined via a convex sum (a weighted sum in which the weights add to unity). We do this as follows:

Assume there are k actions with associated weights u_1, u_2, \dots, u_k and that there are n universal joints in the entire body, where each joint involves three rotational degrees of freedom $[x, y, z]$. Any one of the k actions will use only some subset of these n joints. Let C_i be a vector whose values are 1 for each joint j used by action i and 0 otherwise.

Let $A_i = [A_{i1}, A_{i2}, \dots, A_{in}]$ be a vector representing the value $[x, y, z]$ that is generated by action i for every universal joint j . We obtain the position of universal joint j via $\sum(A_{ij}C_{ij}u_{ij})/\sum(C_{ij}u_{ij})$.

In the next sections we describe the way in which we coordinate the variation over time of the weights of different actions.

2.5 Synchronizing actions

The phases of all the actions are synchronized. For example, when a dancer puppet is walking and we ask her to do a classical fondue turn to the right, she won't begin the turn until she is about to put her weight down on her right foot [figure 6]. This is the only sensible thing to do, since one must begin a fondue turn by stepping into it. This requires that both the walk and the fondue turn are built from expressions that run off the same master clock.

We handle transitions between two actions that we wish to have different tempos via a morphing approach. At the start of the transition, we use the tempo of the first action; at the end, we use the tempo of the second action. During the time of the transition, we continuously vary the speed of the master clock from the first to the second tempo. In this way, the phases of the two actions are always aligned during transitions.

We may also define new actions as extended transitions between two or more other actions. For example, we may morph between the running example above and a standing (or attention) pose in which all joint angles are fixed at zero. When the interpolant is in the range 0.3 to 0.5, this interpolated action becomes a visually realistic walk (to the author's surprise). But a human walking tempo is also 0.3 to 0.5 that of a human running tempo. For this reason, we use the morph transition parameter to modulate the tempo. In this way, a puppet can be made to continually (and realistically) transition from standing still through walking to running, or to anywhere in between.

In a scene with multiple puppets (see section 4 below) each puppet maintains its own individual tempo.

2.6 Dependencies between weights

Let's say that the puppet is walking and we decide to have her do a pirouette. It would make no sense for her to continue walking while she is pirouetting. The mechanism we employ is to build dependencies between weights.

The pirouette weight acts as an inhibitor — as its value rises from zero to one it drives down the effect of the weight that controls such steady state actions as walking. Then as the pirouette weight drops down to zero again the walking weight is allowed to take effect again.

The numerical value of the walking weight is not itself modified. But anything that depends upon it is seen through the filter of the pirouette weight. Conceptually, the pirouette weight "blocks" the walking weight much as the alpha channel of a foreground image blocks a background image during a compositing operation [figure 7].

Note that an action need not involve all joints. Examples include such actions as waving with the left arm, shrugging the shoulders, or scratching one's head. If an action which involves only a subset of the joints blocks another action, then it will only block those joints included in this subset. So we may use this structure to "layer" partial actions. For example, a puppet that is running can be told to wave his hand without breaking his stride.

A small section in the program creates a layering structure for such dependencies. This control is divided into two levels — *states* and *weights*. The state level consists entirely of discrete boolean values. For example, either the puppet "wants" to walk or she does not. The weight level consists of continuous values between zero and one. These are derived by integrating the effect over time of the discrete states [figure 8]. These continuous weights are what go into the convex sum above to drive the puppet. Some of these weights are dependent on others.

For example, if a user directs the puppet to walk, then the discrete walk state turns on and the continuous weight controlling the walking action gradually rises from zero to one. If the user subsequently directs the puppet to perform a pirouette, the weight of the pirouette action gradually rises to one. The discrete walk state continues to stay on, but the continuous weight of the walk is driven down to zero by its dependency on the weight of the pirouette action. When the pirouette state is disabled, the weight of the pirouette action gradually falls to zero and the walking action at the joints gradually reappears.

The dependencies between weights are implemented by a sequence of conditional expressions. This approach is similar in spirit to Brooks' subsumption architecture (Bro86) for walking robots in which more immediate goals (eg. don't fall over) block out longer term goals (eg. "walk to the edge of the table").

2.7 Transition times

Each user specified action starts the rise of some scalar weight from zero to one via an S shaped ramp. We have found that the only tuning needed for controlling the shape of any given transition is a single scalar value that specifies the duration of the transition from

zero up to one or back down again. This is specified in seconds, not frames. Behavior should not change with frame rate! In order to effect this, we use the actual system clock, not a frame counter, to time transitions. We also use the system clock to drive the signal sources described above in section 2.3.

Some transitions look better when they are fast, and others look better when slow. It is surprising how much expressiveness one can achieve by tuning these transition times. For example, when the dancer puppet performs the action "put hands on hips indignantly and look at the camera," she puts her hands on her hips first, and only then, a beat later, does she turn to look at the viewer [figure 9]. Then when she goes from this state into the slow dance, she continues to look at the viewer for a second or so, even while she's already dancing, and her body is turning away.

We conjecture that this behavior looks correct because fixing one's gaze on another person is a more explicit emotional signifier than is changing one's bodily activity, and therefore should happen more slowly. In this case, even when she is beginning to dance, we still want the dancer to convey the reminder that she was annoyed at us just a moment ago.

We believe that the use of different transition times for various parts of a gesture is a very powerful means for conveying subtle impressions of intention. Although our approach to this is currently ad hoc, we hope that experimentation of this kind can lead to a set of useful rules for understanding of human body language. Related work in the role of emotion and communication in gesture is found in the chapter by Calvin and Morevic in (BBZ91).

2.8 Non-hierarchical motions

The puppet will always conform to certain simple constraints no matter what the forward kinematics specify. Here are the key constraints:

- the foot on the ground propels the puppet
- the supporting foot must be at floor level
- obstacles are avoided by turning away from them
- the head won't turn all the way around backwards

Each of these constraints is imposed by a few lines within the code that models the body. For example, to propel the puppet from the foot, we detect the lowest foot. We measure how far this foot has moved since the previous frame. We then add this to a cumulative displacement, and apply a positional offset to the rendered body equal to the opposite of this displacement. The effect is that the lower foot always stays in place, and propels the body.

Also, whenever we compute each new total foot position, we average in half of the previous total just for the y (vertical) component. The effect of this is to always keep the supporting foot level with the ground.

Object avoidance is done as follows. Each wall emits a repulsive force vector which increases near the wall. We sum all of these vectors. If the puppet walks into such a vector field and is angled off to the left (or right) of facing the wall, then we give her a tendency to turn more to the left (or right). When this is tuned properly, she just avoids walls, and so we don't have to worry about collisions. In the more general case, we would put a similar repulsive vector field around any object we want her to avoid, as well as an attractor field at each open doorway. This would act as a variety of remote compliance to help her find her way in.

We can make the puppet look at the camera just by turning her Neck joint. Actually, she can look at any aim point in the scene. But this is not desirable when the puppet's body is facing directly opposite from this direction. To avoid complete backward head turns, we add a constraint into the neck turning joint. A dot product of the body's forward position and the desired aim direction is calculated. As the value of this dot product drops from 1.0 down to -1.0 , we continually lessen the factor by which we influence the head to turn in the aim direction. We found through trial and error that we get the most natural results when this factor reaches zero at a dot product value of -0.6 .

The visual effect is that as the puppet turns away from us, she holds our gaze for a bit and then gradually ignores us as she continues to turn further away. Then as she continues turning, she eventually locks her gaze with us again on the other side, by turning her head over her other shoulder [figure 10].

The above constraints constitute all of the physics built into the model other than the natural constraints imposed by the forward kinematics itself (ie. that the limbs never fly apart).

2.9 Shifting body parts

In order to achieve real time performance, it is important to limit the elaboration of puppet geometry (see next section). Yet we do not wish to sacrifice the appearance of human form. To attain a natural appearance without using large numbers of body parts, we shift body parts around as joints flex, in order to keep the visual appearance of human form for all body angles.

For example, the thigh consists of three intersecting ellipsoids, one for the main thigh mass, a second for the muscle in back of the thigh, and a third for the muscle high up and inside the thigh. When the puppet bends the thigh very far backward about the hip (such as in a fondue turn [figure 6]) the two front ellipsoids have a tendency to separate from the puppet's pelvis. We compensate as follows. As the hip joint bends backwards, we slide these ellipsoids *down* the thigh (away from the hip) in linear proportion to the degree of bending [figure 11]. We provide similar sliding mechanisms at all body parts where such compensation is needed.

2.10 User interaction

User interaction is quite simple. The user only needs to control the discrete states of the puppet. Currently this is handled by a panel of buttons [figure 12]. All continuous behavior is automatically derived by the system through integration over time, as previously described.

3 Implementation

In our current implementation, all parts are rendered as polygonal mesh approximations of 50 ellipsoids. The simulation has been run on an SGI Indigo II m (at 7.5 frames/sec) or an Indigo 2 (at 1.5 frames/sec) and calls the SGI GL library for rendering.

It also runs efficiently on any UNIX or i86 based LINUX machine, but for this implementation the figure can be rendered only in silhouette. In this case rendering is done by computing the silhouette ellipse for each ellipsoid, and then doing a software scan conversion [figure 13]. Since this is a silhouette rendering, front to back ordering does not need to be taken into account. Using this method, the dancer runs at 6 frames per second on a i86/DX66 processor.

The button panel is implemented via a small stand-alone Tcl/tk program. Communication with the this program is done through a two-way asyn pipe.

4 Ongoing and Future work

We are looking at the combination of these procedural techniques with motion capture. The research question here is how to analyze motion capture of walk cycles or gestures in order to convert them into a form compatible with the procedural synthesis techniques. Our approach is to align the natural cycles of walks and other rhythmic motions so that they can be blended together.

We also are beginning to study group interactions between these simulated puppets. This research is focused on situations in which people communicate richly through body language, such as parties, bar scenes, meetings.

Because all control of a puppet's state is discrete, knowledge of actions and transitions between two or more interacting puppets can be done by exchanging state tokens. If a character knows the state of another character, then for non contact pairwise interactions it suffices to know only the position and facing direction of the other character. This method of communication is fast and compact, and scales up gracefully in simulations with large numbers of puppets.

Using this approach, we make the characters "press each others' buttons". For example, when two characters are engaged in conversation, they tend not to simultaneously talk at once (although they occasionally do) and they also tend to avoid long collective silences. When a third character walks up, the behavior of the other two shifts, depending upon the status of the newcomer and how each of the other two feels about him/her.

A related notion that we will explore is peripheral attention. For example, suppose a man and a woman are engaged in conversation, and another man appears in the line of vision of the first. How would one show the fact that the effect of the woman's attention involuntarily drifting toward the other man, even though her intention is to maintain the conversation? Similarly, how would one show the shift in each man's attitude when the woman's behavior is noticed? The first man might begin to talk more frequently, the second might drift toward the conversation.

In recent work, we have developed methods of running these group simulations on multi-processors and across multiple networked computers. This is done over a network of UNIX workstations as follows. Each actor is a separate program which communicates through its standard input and standard output. A supervisory rendering process opens up a two-way read/write pipe to each actor. Each actor may be invoked via a remote shell, so that it need not be on the same workstation as the renderer. To send messages, actors print commands to their standard output which are parsed and executed at each frame by the supervisor program. If one actor wants to send a message to another, then it prints a wrapper command. This wrapper command instructs the supervisor program to print a message command string to the standard input of the recipient actor. The recipient then parses and executes this message.

This approach makes it quite easy to allow different kinds of actors to each respond in the most appropriate way to a given message. The Camera is an actor which possesses behavior like any other. For example, in our current system Actor1 can send the messages

```
{ my_location "look_here" } "Camera" send_message  
{ my_location "look_here" } "Actor2" send_message
```

where *my_location* is the current x, y, z position of Actor1. Note that different recipients are free to interpret any message as they see fit. For example, the Camera actor generally averages all "look_here" requests, so that it keeps all attention-seeking actors in its range of vision. In contrast, if several "look_here" requests are sent to a human actor, it will generally honor only one of them. As a result, an actor will adjust his/her gaze to track only the message sender of greatest interest. This provides a simple object-oriented message capability, with overloading of methods based on the type of the recipient.

In addition, we are exploring immersive interactions using projector screens and position sensors, so that real people can interact with these characters, which are digitally composited into miniature models of interior spaces. Within this experimental laboratory, we explore questions of how to convey peripheral awareness, approach/avoidance, paying attention, "listening," etc. This is in the spirit of the recent *Muc* project of Macs et al. at MIT (Macs93). We are particularly interested in immersive scenarios involving two or more projection screens, in order to see to what extent simulated body language will help to convey the impression of various competing social or attention-getting activities.

We are also studying the semantics of the discrete state transitions that visually represent shifts in attitude and attention. We are particularly interested in determining to what extent can we encode merely the rhythm of interpersonal interaction, in order to convey

the impression of social complexity. For example, could one structure entire narratives in this manner?

5 Conclusions

Using ideas from procedural texture synthesis, we are able to create remarkably lifelike, responsively animated characters in real time. By conveying just the “texture” of motion, we are able to avoid computation-intensive dynamics and constraint solvers. We believe these techniques have the potential to have a large impact on computer Role-Playing Games, simulated conferences, clip animation, graphical front ends for MUDs, and synthetic performances.

6 Acknowledgements

I would like to thank Athomias Goldberg for production support on this paper, and in particular for the illustrations, as well as on the work itself. I would also like to thank Cynthia Allen, David Bacon, Troy Downing, Mehmet Karaul, Tom Fiskawy, Kuochen Lin, Jon Meyer, and Jack Schwartz for all their help and encouragement. Ben Bederson, Bruce Naylor, and Silicon Graphics Inc. have provided hardware assistance for this research. Thanks as well to Marcelo Zuffo, Roseli Lopez, and the folks down at the University of Sao Paulo for all their support. And *multo obrigado* to Emi, who inspires the dance.

7 References

- Norman I. Badler, Brian A. Barsky, and David Zeltzer. *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*. Morgan Kaufmann Publishers, San Mateo, CA, 1991.
- N.I. Badler, C. Phillips, and B.E. Webber. *Simulating Humans: Computer Graphics, Animation, and Control*. Oxford University Press, 1993.
- R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.
- J.E. Chadwick, D.R. Haumann, and R.E. Parent. Layered construction for deformable animated characters. *Computer Graphics (SIGGRAPH 89 Proceedings)*, 23(3):243–252, 1989.
- D. Lbert and et al. *Texturing and Modeling: A Procedural Approach*. Academic Press, London, 1991.
- D. Getlernter. *Mirror Worlds*. Oxford University Press, 1992.
- M. Girard and A.A. Maciejewski. Computational modeling for the computer animation of legged figures. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 20(3):263–270, 1985.

- P. Maes. The mit alive project. *Computer Graphics (SIGGRAPH 93 Proceedings)* 1993
- Claudia I. Morawetz and Thomas W. Calvert. Goal directed human animation of multiple movements. *Proc. Graphics Interface* pages 60-67. 1990
- K. Perlin. An image synthesizer. *Computer Graphics (SIGGRAPH 85) Proceedings* 19(3) 287-293. 1985
- K. Perlin. Danse interactif. *Computer Graphics (SIGGRAPH 91 Proceedings)* 25(3) 1991
- M. Rubert and et al. *Legged Robots That Balance*. MIT press. 1986
- SGI. *SGI Programmers Manual*. Silicon Graphics Incorporated. Mountunview. 1991
- Karl Sims. Evolving virtual creatures. *Computer Graphics (SIGGRAPH 91 Proceedings)* 25(3) 15-22. 1991
- Neal Stephenson. *Snow Crash*. Bantam Doubleday. New York. 1992

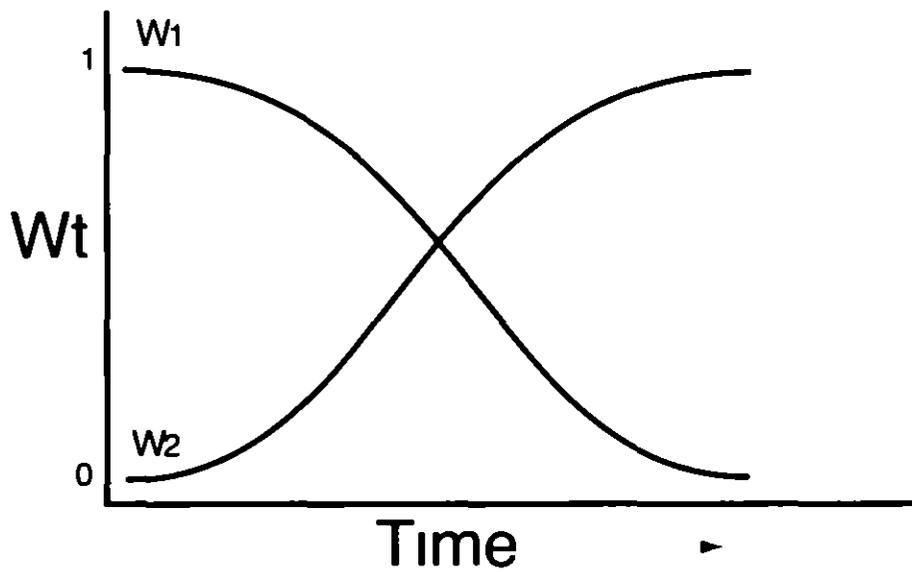


figure 1

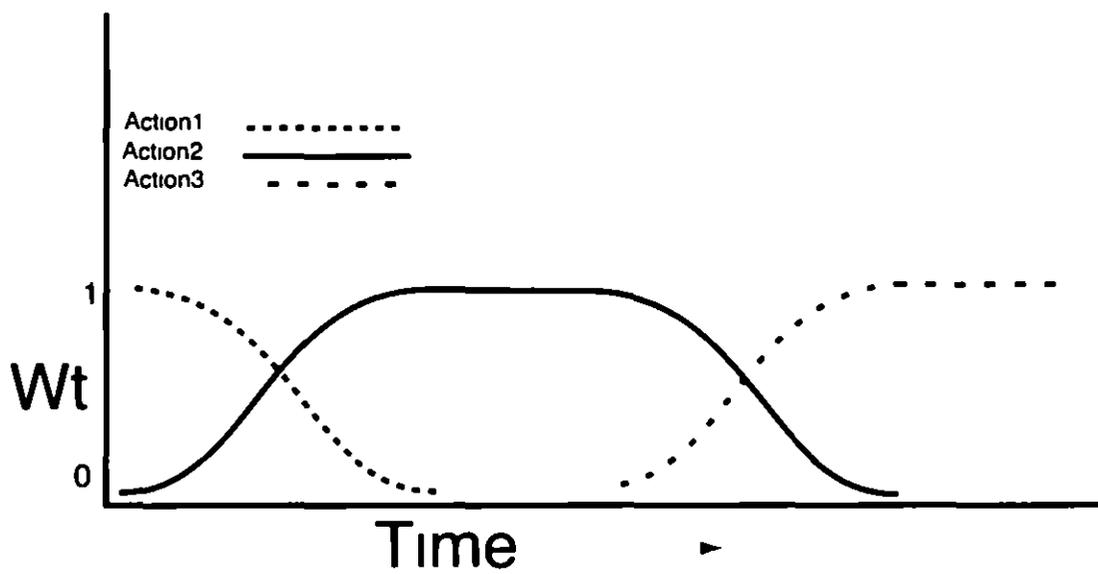
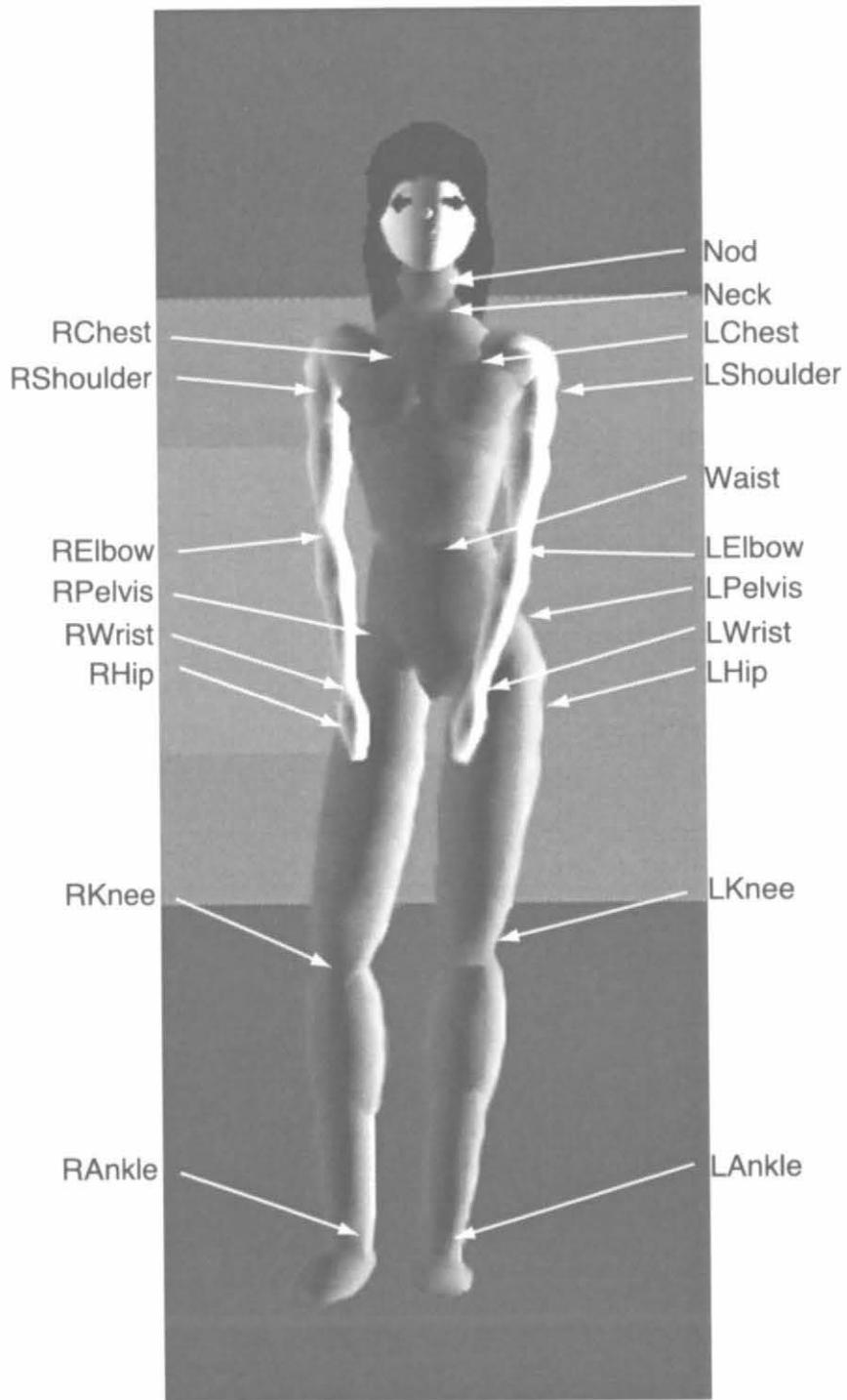
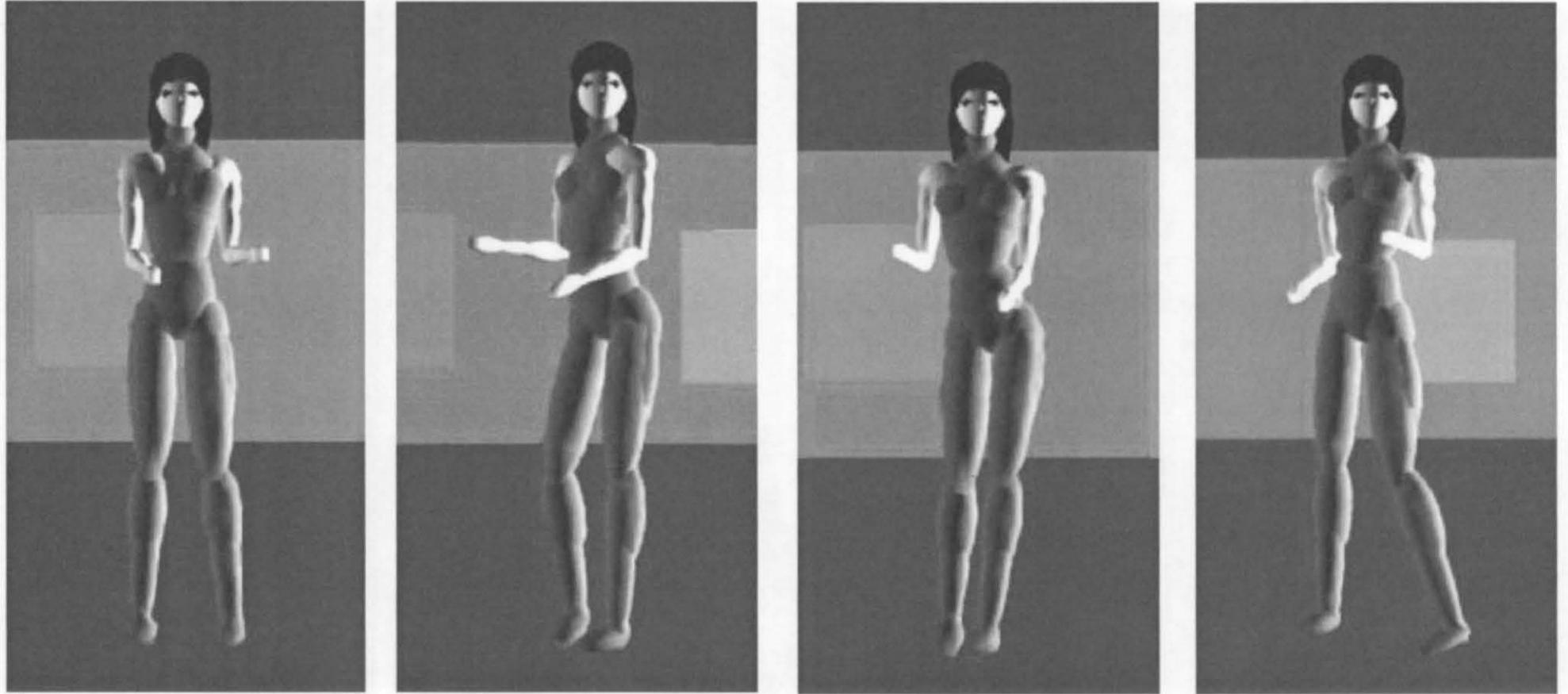


figure 2



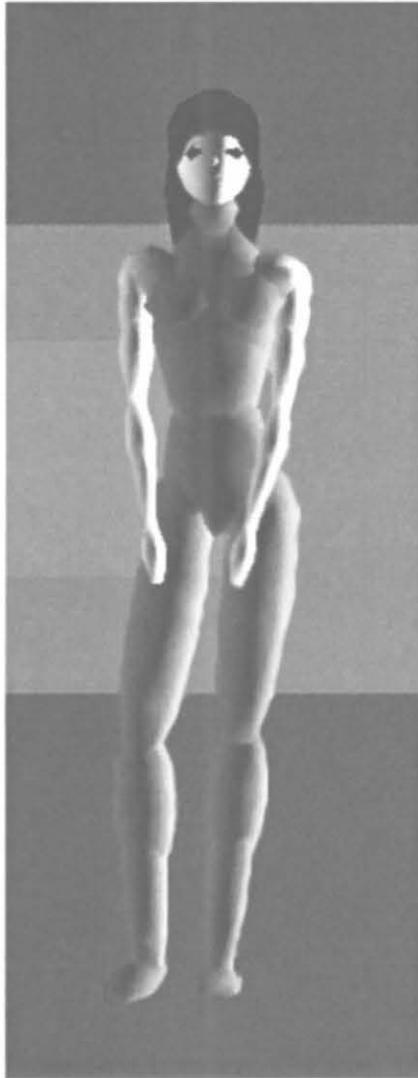
19 Universal Joints

figure 3



Four Phases of a Rhumba

figure 4



Standing

figure 5



Transition into a fondué turn

figure 6

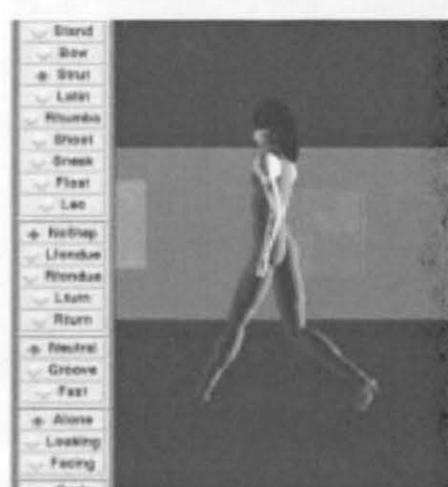
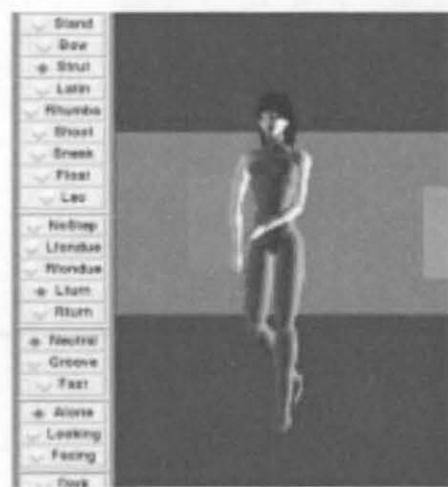
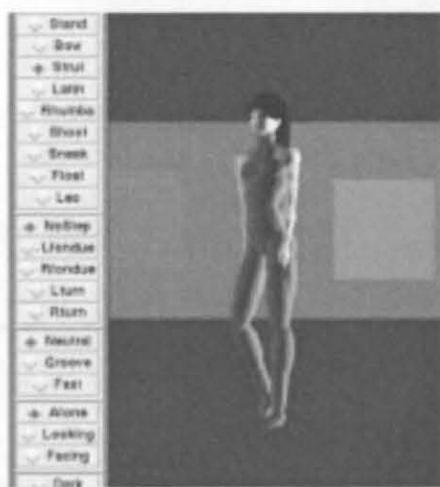


figure 7

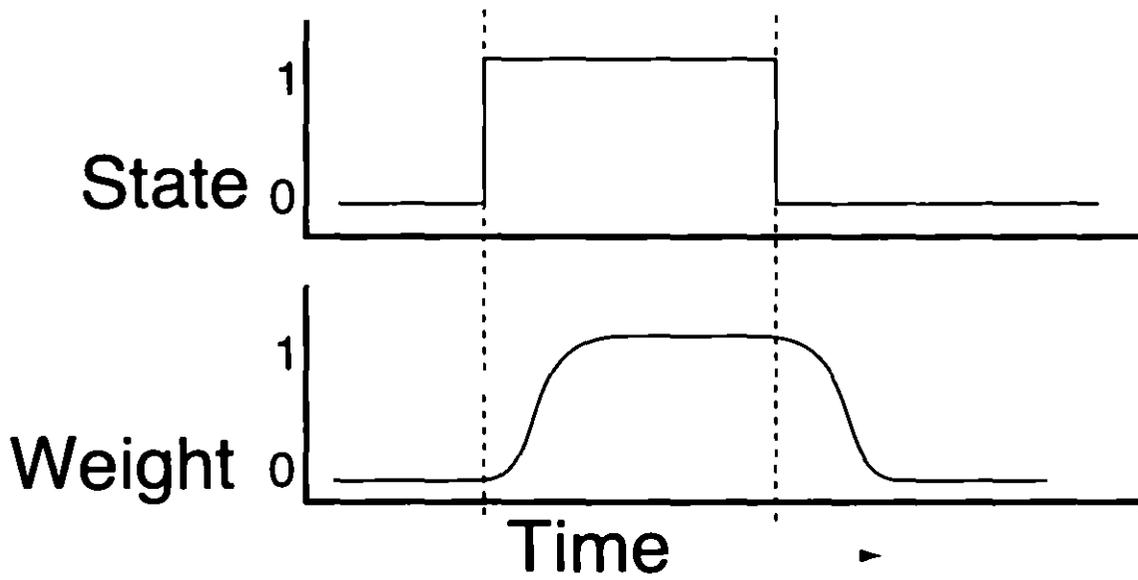


figure 8

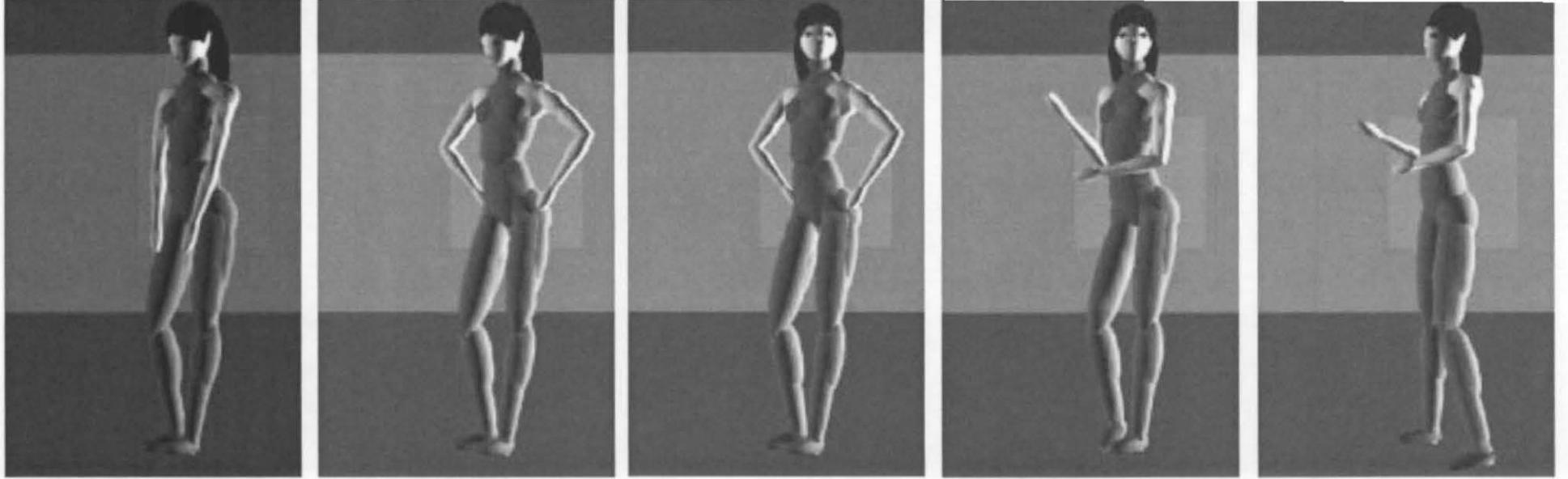


figure 9

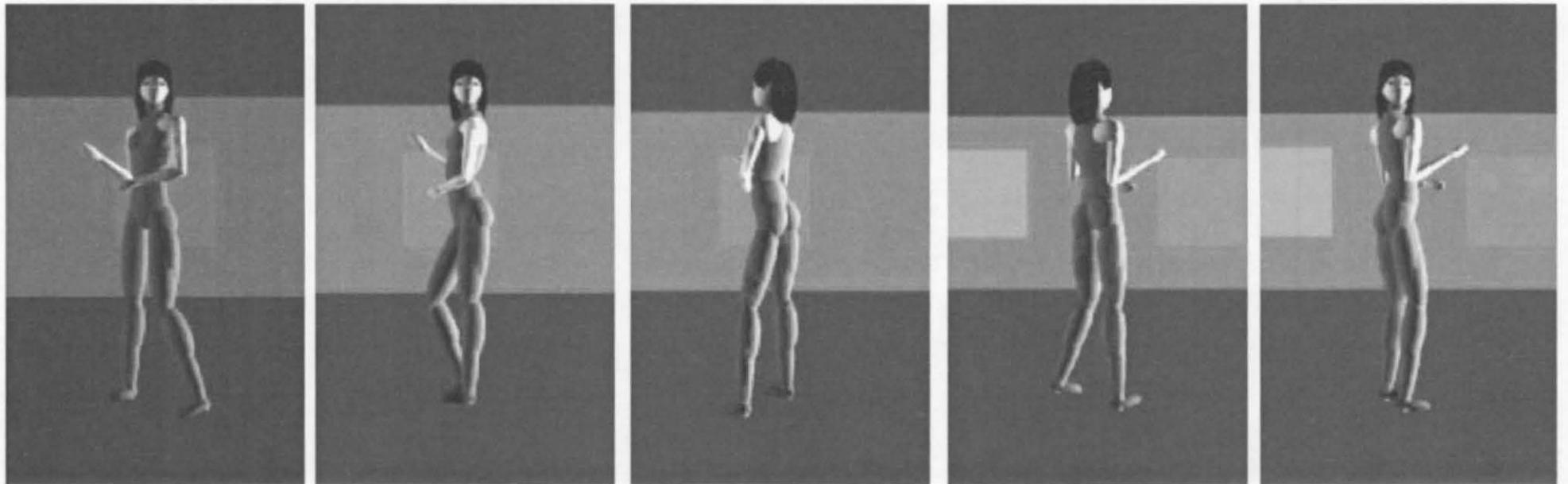


figure 10

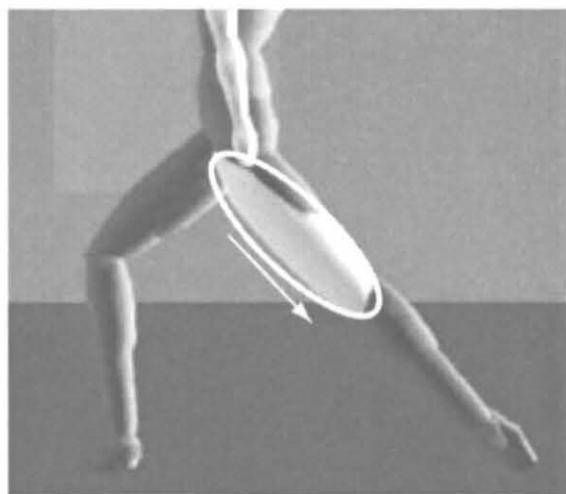
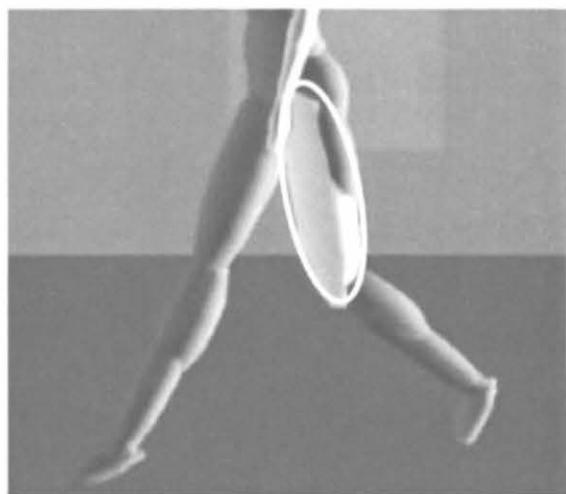


figure 11



figure 12

MIRALab, U. of Geneva

From 2D Photographs to MPEG-4 Compatible Talking-Head

Nadia Magnenat Thalmann

**MIRALab, University of Geneva
24, rue General Dufour
CH 1211 Geneva
Switzerland**

**e-mail: Nadia.Thalmann@cu.unige.ch
fax: +41 22 705 77 80
tel: +41 22 705 77 69
<http://miralabwww.unige.ch/>**

MIRALab, U. of Geneva

Summary

- **Facial reconstruction from two pictures**
- **Real-time face animation**
- **Facial animation in the MPEG-4 standard**

Facial Reconstruction from Two Pictures

- Introduction
- Feature Points Detection
- Modification of a Generic Model
- Texture Mapping
- Results

3

Introduction



4

Feature Points Detection

- **Features on front and side view**
 - eyes, nose, ear, lips, hair and face outlines, etc.
- **Feature Detection**
 - structured snake
 - anchor some points - to keep structure of points
 - color information
 - multiresolution techniques - to get stronger edges
 - interactive correction

5

Modification of a Generic Model

- **Generation of (x, y, z) from (x, y_r) and (y_s, z)**
- **Dirichlet Free Form Deformations (DFFD)**



Generic Model

+



3D Feature lines



Modified Head

6

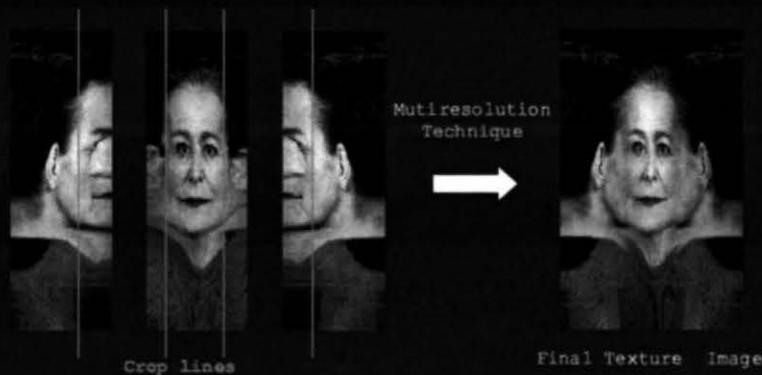
Texture Mapping

- **Texture Generation**
 - One texture image from two images
- **Texture Fitting**
 - Feature points fitting
 - Non-feature points fitting

7

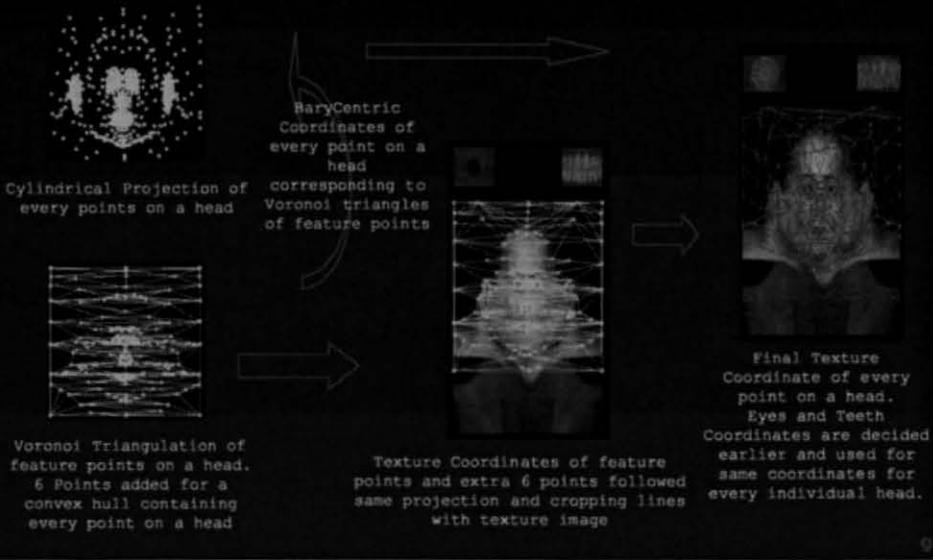
Texture Generation

- **Cylindrical projection**
- **Image mosaic - Multiresolution technique**



8

Texture Fitting



Results



Front Image

+



Side Image

→



A 3D head animated

Results



- **Snapped from a 3D reconstructed head**
- **backsides has proper texture too**
- **Animation structure contained automatically**

11

Results

- **More Examples**



12

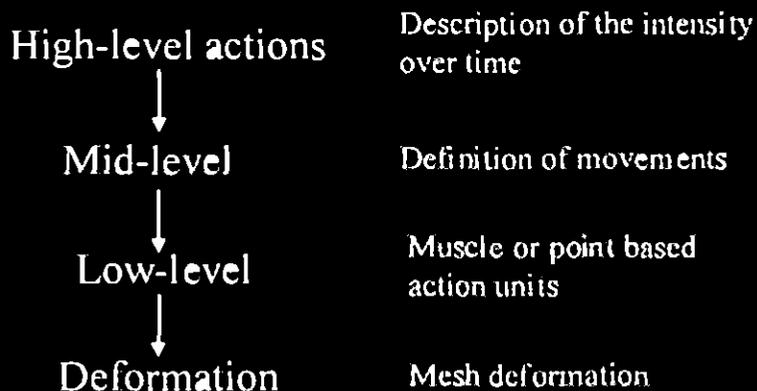
Real-Time Face Animation System

4 levels of decomposition are used to generate our facial animations:

- High-level : Emotions, Sentences, Head movements
- Mid-level : Expressions, Phonemes
- Low-level : MPAs or FAPs
- Deformation-level : RFFD or DFFD

13

Levels hierarchy and effects



14



High-level actions

To allow an animator to work on a task level where she/he can define animation in terms of its inherent sources:

- Emotions**
- Sentences**
- Head Movements**



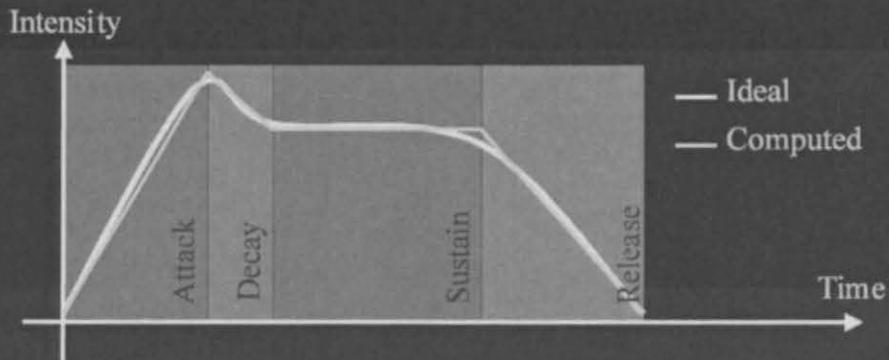
High-level actions

Emotions

- An emotion is interpreted as an evolution of a face over time. Starting from neutral state, passing through a sequence of visible changes, and returning to a neutral state. We use an envelope consisting of 4 stages to generate these variations.**
- Typical emotions could be: Smile, anger, surprise, fear ...**

High-level actions

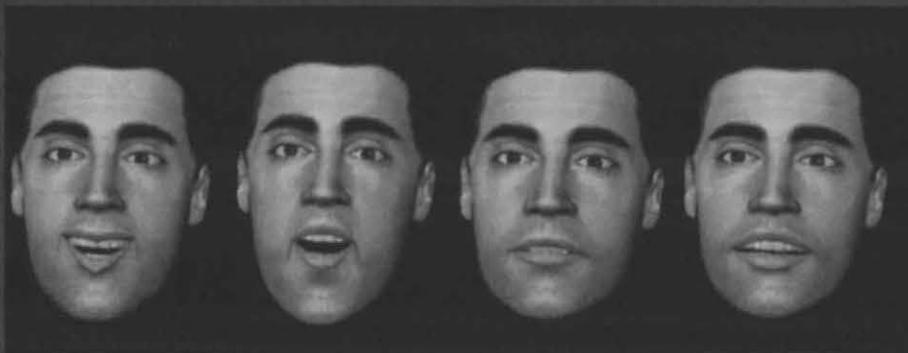
Emotions envelope



17

High-level actions

Emotion samples



Happy

Surprise

Sad

Fear

18

High-level actions

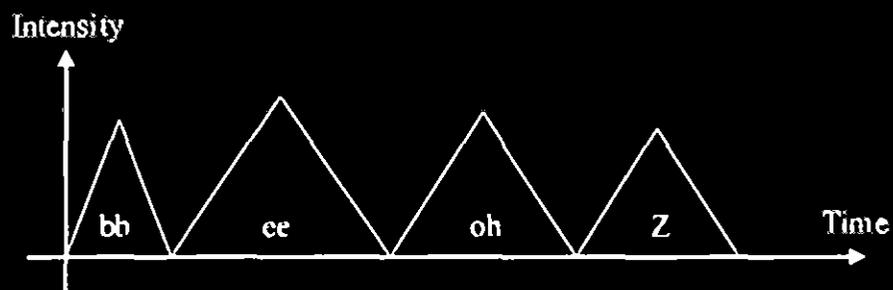
Sentences

- This level has been introduced to allow easy speech animation without having to work on the phoneme level. Each sentence action is a temporal description of a sequence of phoneme.

High-level actions

Sentence envelope

- "because" → 'bb' 'ee' 'oh' 'Z'



High-level actions

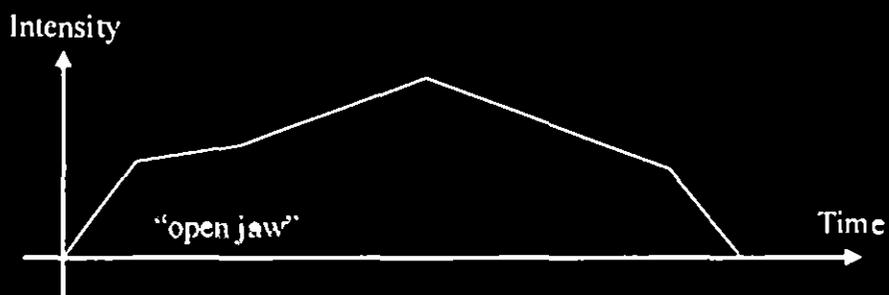
Head movements

- These high-level actions describe the intensity of head movements over time.
- The intensity of the head movement can be fixed at any time.

21

High-level actions

Head movements envelope



22

Mid-level actions

- **Mid-level actions defined as expressions and phonemes are considered as facial snapshot.**
- **They are modulated in time and intensity by the high-level actions.**
- **A facial snapshot consists of a set of low-level action units.**

23

Mid-level actions

Phoneme examples



24

Low-level actions

At this level 2 competitive methods of action units have been implemented:

Minimal Perceptible Action

MPA is a set of basic facial motion parameters based on facial muscles location. They are region based.

Face Animation Parameters

FAP is the MPEG-4 standard for facial animation. It consists of a set of feature points located at strategic location on the face.

25

Low-level actions: MPAs

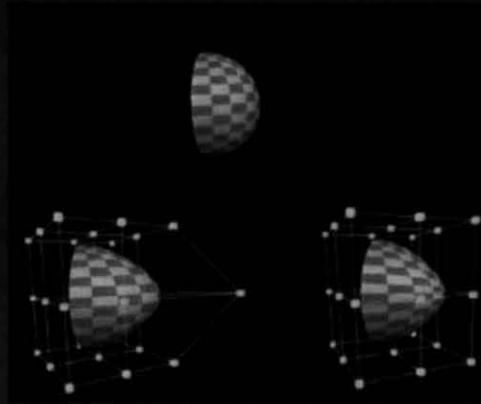
Region
definition
for MPAs



26

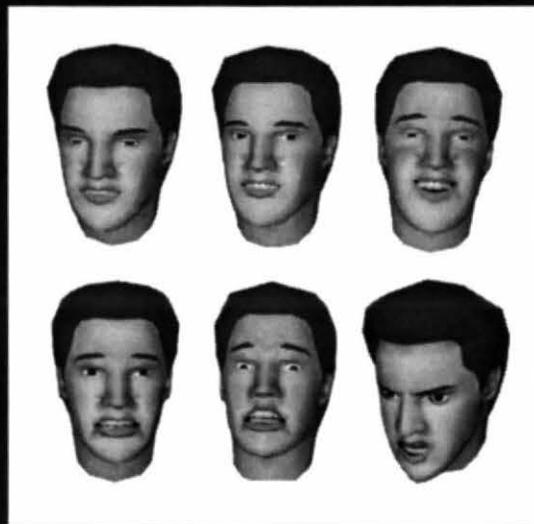
Low-level actions: MPAs

Rational Free Form Deformations



Low-level action: MPAs

Results using MPAs



FaceAgentLib

In order to meet all needs for software developments using face animation, we have compiled a high-level library filling the following requirements:

- easy high-level face animation**
- multi-actor**
- real-time**
- graphic library independency**
- multi-action blending**

29

FaceAgentLib

Functions

- Reads high-levels actions**
- Supports multiple heads**
- Enables duration, intensity and weight settings at action activation**
- Allows action interruption with or without delay**
- Allows reactivation of interrupted action**
- Outputs frames of MPAs or FAPs**

30

FaceAgentLib

Multi-actor animation concept:

- An agent is associated to each actor/face
- Each agent has a updateable list of available actions
- Any of these actions can be activated
- Each activated action has an associated list of array of MPAs
- At each frame, one can get from every agent the result of the blending of every active actions
- The result is an array of MPAs

31

FaceAgentLib

Multiple action blending

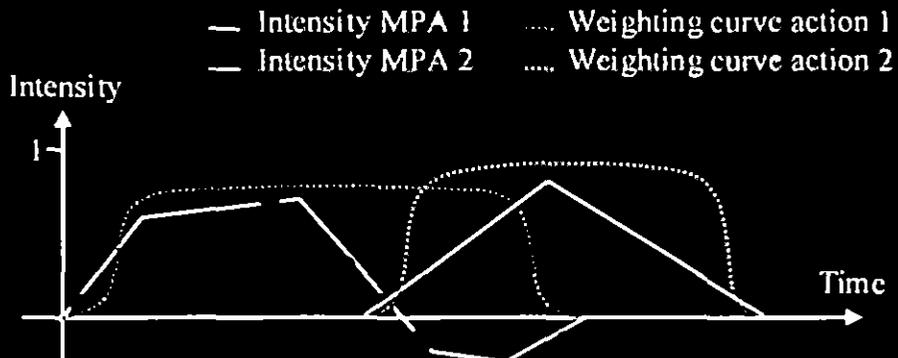
To generate the blending of multiple high-level actions we have to take into account:

- the envelope of each action
- the blending curve of each action
- the list of MPAs used by each action
- the weight of each action

37

FaceAgentLib

Multiple action blending example



33

FaceAgentLib

Multiple Action Blending

For each additional MPA we apply the following formula to get the resulting Intensity:

$$\text{Current_intensity} = \text{Current_intensity} + (\text{Action_weight} * \text{MPA_intensity})$$

34

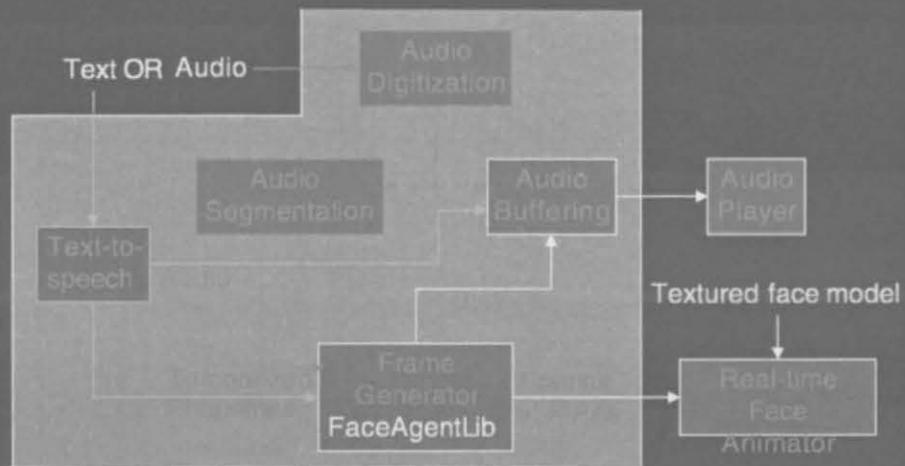
Talking Head

Requirements:

- **input:**
 - face model with texture
 - text or audio voice
- **output:**
 - real-time face animation
 - synchronization of animation with voice audio output

35

Talking Head



36

Talking Head

Snapshots



37

Talking Head

Additional comments

- The use of FaceAgentLib to generate the mouth animation gives the possibility to use all the high-level actions for a complete behavioral animation.
- This ability is used by the "virtual director" interface to control and fully animate multiple actors.

38

4

Virtual Producer

The aim is to provide an “user-friendly” interface dedicated to dialogue between virtual humans.

It is used for the real-time control of virtual humans speech in computer animation sequences in augmented reality.

4

Virtual Producer

Features of the interface:

- multiple virtual human control in real-time**
- Interactive adding/activation of sentences, emotions and cameras**
- high-level control allow to link sentences with emotions and cameras**

Virtual Producer

Final result improvement

- "live recording" allows post production (useful for precise synchronization)
- interface and 3D environment can be distributed on different machines

41

Virtual Producer

Example: interface for two virtual actors

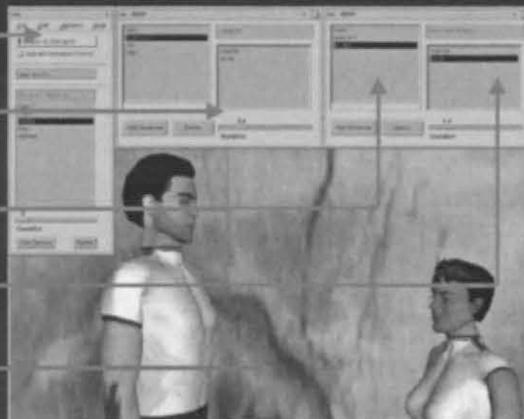
Virtual Cameras
Interface

Speech Interface for
virtual actor 1

Sentences of virtual
actor 2

Emotions of virtual
actor 2

3D environment



42

Facial Animation in the MPEG-4 standard

- **Introduction to MPEG-4**
- **Face and Body Animation (FBA)**
- **Face Animation Parameters (FAP)**
- **FAP Interpretation @ MIRALab**
- **Face Definition Parameters (FDP)**
- **FDP Interpretation @ MIRALab**

Introduction to MPEG-4

MPEG-1,2:
frame-based video



MPEG-4: AV objects



- **Set of dynamic AV objects**
- **Composition on decoder**
- **Interaction**
- **Synthetic objects**

Synthetic objects in MPEG-4

- **3D scene hierarchy (VRML compatible)**
- **2D text and graphics**
- **Synthetic audio**
- **Text to Speech**
- **Face and Body objects**

45

Face and Body Animation (FBA)

- **Efficient representation of shape and movement of human bodies and faces**
- **Applications**
 - **Communication**
 - **Entertainment**
 - **Ergonomics**
 - **Medicine**
 - **etc.**



46



Face and Body Animation (FBA)

- **Definition parameters for shape**
- **Animation parameters for movement**

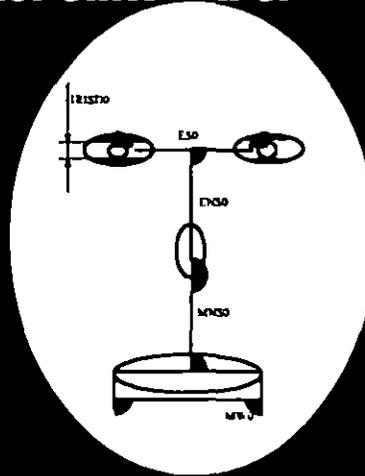


Facial Animation Parameter set

- **Complete**
 - **Set of basic movements to model any expression**
 - **High level parameters for visemes and expressions**
- **Efficient**
 - **No redundancy**
 - **2 Kbit/sec for full facial animation @ 25Hz**
- **Model independent**
 - **Normalized coordinates with Facial Animation Parameter Units (FAPU)**

Facial Animation Parameter Units (FAPU)

- **Normalized parameters**
 - Each animation parameter expressed in terms of a related FAPU
 - e.g. mouth stretch expressed in units proportional to neutral mouth width



49

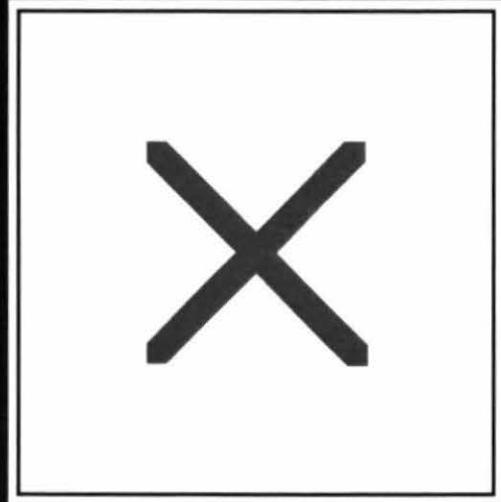
FAP Interpretation @ MIRALab

- **Face model**
 - Polygon mesh
 - Facial regions defined as subsets of polygons (mouth, eyes etc.)
- **Animation**
 - Free Form Deformations (FFD)
 - Each parameter activates one or more FFDs on a region

50

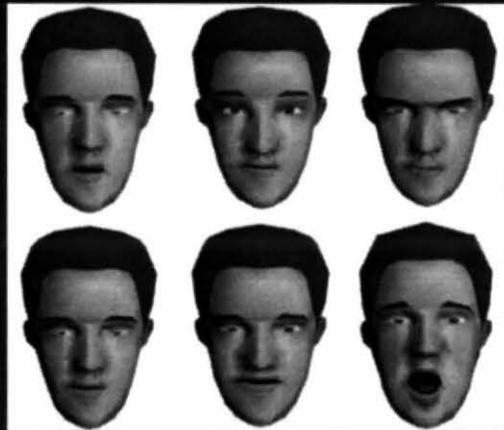
FAP interpretation examples

- Different facial models interpreting same FAPs



FAP interpretation examples

- FAP animation snapshots (Marco sequence)



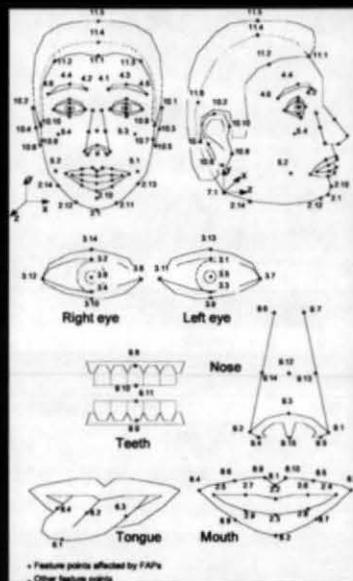
Facial Definition Parameter set

- **Optional**
- **Scalable**
 - **Face Feature Points**
 - **Texture Coordinates for Feature Points (optional)**
 - **Face Scene Graph (optional)**
 - **Face Animation Table (FAT) (optional)**

53

Face Feature Points

- **Normalize animation parameters**
- **Find feature correspondence in different faces**
- **Roughly define shape**



54



Face Feature Points

- **Applying Feature Points and texture to reshape a facial model**
- **IMAGE**



Face Calibration Model

- **Contained in Face Scene Graph**
- **A complete polygon mesh face model**
- **Used to define the shape of the face**
- **The face is reshaped by fitting to the model**

Face Animation Table (FAT)

- **Used to define the animation behavior of the face model defined in Face Scene Graph**
- **Action of each Animation Parameter defined on the model as piece-wise linear movement of vertices**

57

FDP interpretation @ MIRALab

- **Face Feature Points**
 - **DFFD**
- **Texture**
- **Face Calibration Model**

58

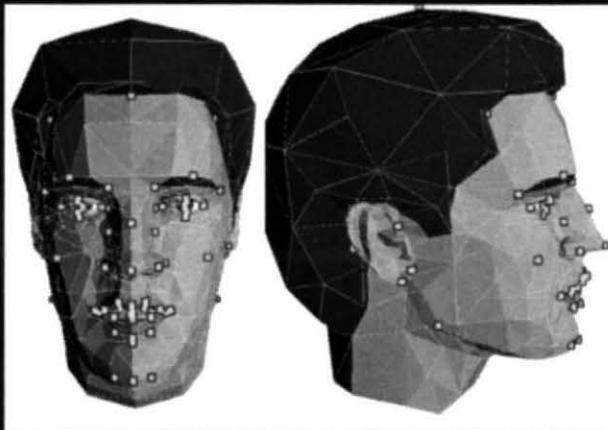
Basic steps for face reconstruction

- Definition of a generic head
- Cylindrical projection of the generic head
- Delaunay triangulation and barycentric coordinates extraction
- Texture coordinates computation using FDP 2D texture coordinates
- Extraction of the 3D position of the non-extracted control points
- DFFD applied on the generic head using FDP coordinates
- Final rendering using the modified generic head and the texture

59

Generic head definition

**Generic head
and MPEG-4
feature
points**



60

Mesh deformation using DFFD

Dirichlet Free Form Deformation characteristics:

- allow a volume deformation using control points
- keep the surface continuity
- uses a Dirichlet diagram to compute the Sibson's local coordinates for the non-feature points interpolation.

61

Mesh deformation using DFFD

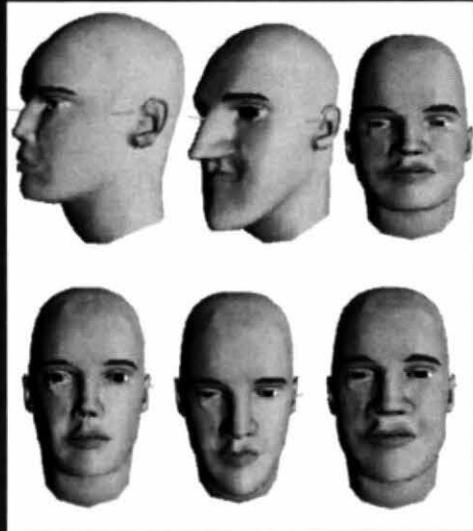
Chin deformation using DFFD



62

Mesh deformation using DFFD

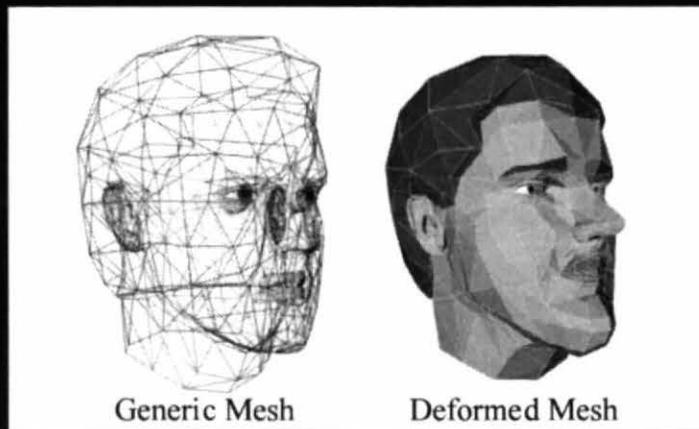
Head samples
deformed with DFFD



63

Face reconstruction

Mesh deformation result



64

Face reconstruction

Results using MPEG-4 datas



1 Introduction

To clone a face has attracted people for a long time in the real world and virtual world. With the growing power of computer speed and multimedia ability, people would like to have their counterpart in virtual world and animate it and utilize it for communication in an efficient way.

However, until now to get a realistic facial reconstruction in commercial equipment and very detailed matched face has been considered as a growing research area. We show our main three methods to generate a clone from a given input, such as one or several random pictures of a person, orthogonal pair of front and side views of a person, or range data obtained with more sophisticated equipment or a rather complicated algorithm. Table 1 classifies these three inputs and results.

Input	Equipment	Method	Result	Time
Pictures (unorganized)	Mono camera	Manual using user friendly designing software	Detailed matching with human eyes	days/weeks
Pictures (organized)	Mono camera	Automatic methods using Feature detection and generic model modification	Rough matching	minutes
Range data (detailed shape, no animation structure)	Laser scanner/ Light stripper/ Stereoscopic camera + extra	Automatic methods using Feature detection and generic model modification and fine adaptation	Detailed matching	minutes

Table 1 Three possible ways to get a cloned face for animation in virtual world

2 Unorganized picture data

There are many ways to make a clone of a person who is not available to have pictures with special purpose for instance Marilyn Monroe. Sometimes only one picture is available or several pictures taken at different times and places which have great difficulty to get 3D data without help from human eyes. To get a detailed shape, we need to do a time-consuming manual job with help of user friendly graphical interface.

Plaster Model Magnenat, Thalmann et al. [15] used plaster models in real world and selected facets and vertices marking on the models which are photographed from various angles to be digitized. Then all the 2D coordinates are combined to produce 3D data of the face. Here the reconstruction approach requires a

mesh drawn on the face and is time-consuming, but can obtain high resolution in any interested area. Pixar's new animation character Geri [1] is sculpted, which was then digitized and used as a basis for creating the 3D model.

Interactive Deformation With one or several pictures, we use our *Sculptor* software [18] dedicated to the modeling of 3D objects. The sculpting approach is based on local and global geometric deformations. Adding, deleting, modifying, assembling triangle meshes are the basic features provided by *Sculptor*. Real-time deformations and manipulation of the surface gives the designers the same facilities as with real clay or wax sculpting. There are two ways to create a face. One way to start from a template head, this therefore accelerates the creation process. The second method from scratch, the designer can model half of the head

and use a symmetric copy for the other half. At the end, small changes should

be made on the whole head because asymmetric faces look more realistic.



Figure 2-1 Head creation from a template in *Sculptor* and *TextureFit* program. The upper left image is a photo of the real terra-cotta soldier. The upper middle image is the texture created from the photo. The other objects are snapshots from the 3D model.

Interactive Texture Mapping Texture mapping is a well-known low-cost method in computer graphics for improving the quality of virtual objects by applying real images onto them. For virtual humans, the texture can add a grain to the skin, including the color details like color variation for the hair and mouth. These features require correlation between the image and the 3D object. A simple projection is not always sufficient to realize this correlation: the object, which is designed by hand, can be slightly different from the real image. Therefore an interactive fitting of the texture is required. A program allowing the fitting of the texture according to features of the 3D object is called *TextureFit* [2]. This enables the designer to interactively select a few 3D points on the object. These 3D

points are then projected onto the 2D image. The projection can be chosen and set interactively, hence the designer is able to adjust these projected points to their correct position on the image. Figure 2-1 shows the input and resulted head produced using *Sculptor* and *TextureFit*.

Animation Structure To simulate the effects of muscle actions on the skin of virtual human face, specific regions are defined on the mesh corresponding to the anatomical regions where a muscle is desired. A control lattice is then defined on the region of interest while modeling is processed. This method is necessary when a person is not available at that specific time & place, and specially when we want to generate a character in imagination. The result depends on a designer's

artistic sense and usually it required at least days and weeks time. To give an animation structure is also another deal

since every generated character has different points and faces structures.

3 Organized Picture data

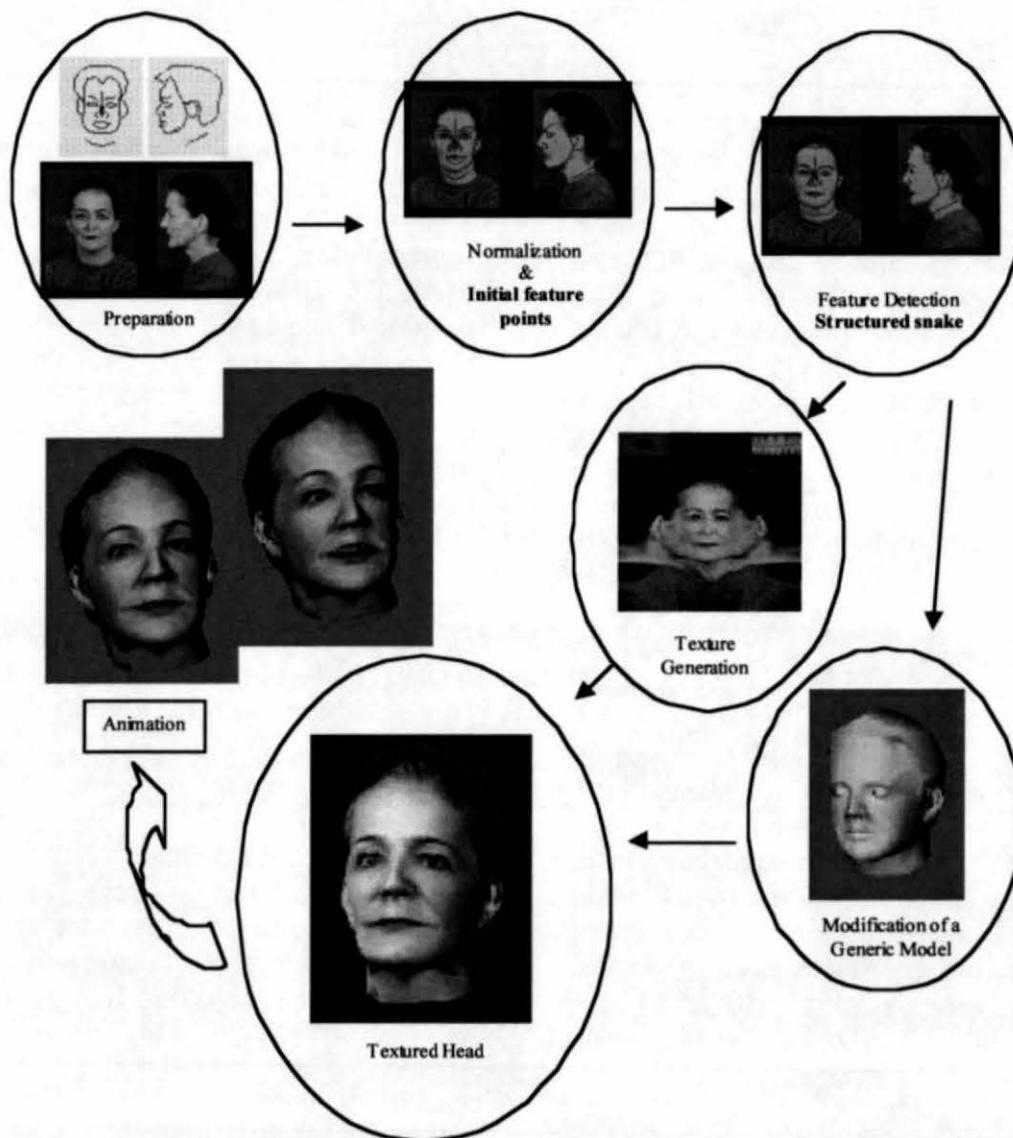


Figure 3-1 Overall flow for 3D-head reconstruction from two orthogonal pictures.

There are faster approaches to reconstruct a face shape from few pictures of a face. In this method, a generic model with animation structure in 3D is provided in advance, and a limited number of feature points, which are the most characteristic points to recognize people, detected either automatically or interactively on the two (or more) orthogonal pictures, and the other points on the generic

model are modified by a special function. Then 3D points are calculated by just combining several 2D coordinates.

Kurihara and Arai [12], Akimoto et al. [5], and Ip and Yin [9] use an interactive method or automatic method to detect feature points and modify a generic model. Each of them has some drawbacks such as too few points to guarantee appropriate shape

from a very different generic head or accurate texture fitting, or too much loose automatic methods like simple filtering and texture image generation using simple linear interpolation blending. We present our approach to reconstruct a real face from two orthogonal views. In this reconstruction, feature points are extracted from front and side views. Reconstruction of a shape may not require high accuracy in some special cases thanks to the texture mapping. However, for animation, we need a process of texture fitting which can ensure positional correspondence of features in the model and the texture image. The reliability of texture fitting is based on the number of feature points. To get better result we need more points at the right place. If an automatic method is not robust enough, it is better to use interactive way. See Figure 3.1

3.1 Preparation & Normalization

First we prepare two 2D wire frames composed of feature points with predefined relation for front and side views. The frames are designed to be used as an initial position for the snake method later on. Then we take pictures from front and side views of a head. The picture is taken with maximum resolution and the face is in neutral expression and pose.

To make the head heights of side and front views the same, we measure heights of them, and choose one point from each view for matching them with corresponding points in prepared frame. As an example we select the highest point on a front face and the top of the nose on a side face. Then we use transformation (scaling and translation) to bring the pictures to the wire frame coordinate, overlaying frames on pictures.

3.2 Feature Detection

We provide automatic feature points extraction method with an interface for interactive correction if and when needed. We consider hair outline and face outline and some interior points such as eyes, nose, lips, eyebrows and ears as feature points. There are methods to detect them just using special background information and predefined threshold [5][9] and then use an edge detection method and apply threshold again. Also image segmentation by clustering method is used [5]. However, it is not very reliable since the boundary between hair and face and chin lines are not easy to detect in many cases. Moreover, color thresholding is too sensitive depending on each individual's face image and therefore requires many trials and experiments. We use a structured snake, which has functionality to keep the structure of contours. It does not depend on the background color much and is more robust than simple thresholding method.

3.2.1 Structured Snake

First developed by Kass et al [11] active contour method, so called as snakes, is widely used to fit a contour on a given image. This allows the fitting of the boundary points of maximum contrast close to the already-defined rough contour. To get correspondence between points from pictures and points on a generic model which has a defined number, a snake is a good candidate. Above the conventional snake, we add three more functions. First, we move few points to the corresponding position interactively, and anchor them. It helps later to keep the structure of points when snakes are involved and is also useful to get more reliable result when the edge we would like to detect is not very strong. We then use color

blending first for a special area so that it can be attracted by a special color [6] When the color is not very helpful and Sobel operator is not enough to get good edge detection we use a multiresolution technique [7] We can insert some more points which are not visible but work as a member of snake to keep non uniform interval between visible points of the snake

We have several parameters such as elasticity, rigidity image potential and time step, to manage the movement of snake As our contours are modeled as polylines we use a discrete snake model with elastic rigid forces and image force acting on each pixel for color interest We define different sets of parameters for hair and face according to their color characteristic To adjust the snake on points of strong contrast we consider

$$F_{ext} = n_i \cdot \nabla E(v_i) \quad (1)$$

where n_i is the normal to the curve at the node i whose position is v_i and is given by

$$E(v_i) = |\nabla I(v_i)|^2 \quad (2)$$

where $I(v_i)$ represents the image itself To estimate the gradient of the image we use the Sobel operator Blending of the different color channels is changed to alter the snake's color channel sensitivity For instance we can make snakes sensitive to the excess of dark brown color for hair Also we use clamping function to emphasize special interesting range of color Snakes are useful in many circumstances particularly in the presence of high contrast The color blending and clamping function depend on the individual

Sobel operator does not always provide strong edge for some areas, for instance, chin lines In such cases we employ multiresolution approach [7] to obtain strong edges It has two main operators REDUCE with Gaussian operator and EXPAND The subtraction produces an image resembling the result after Laplacian operator applied on the image

processing More times the REDUCE operator is applied stronger are the edges

3.3 Modifying a Generic Model

We produce 3D points from two 2D points on frames with predefined relation between points on a front view and on a side view Some points have x, y_s, y_f, z so we take y_s, y_f or average of y_s and y_f for y coordinate (subscripts s and f mean side and front view) Some others have only x, y_f and others x, y_s Using predefined relation from a typical face, we get 3D position (x, y, z) We modify non-feature points with some distance-related functions spring mass model, or FFD method Here we employ Dirichlet Free Form Deformations (DFFD) to move other points according to feature points

3.3.1 DFFD

Distance-related functions have been employed by many researchers [5][9][12] to calculate displacement of non-feature points related to feature points detected We propose to use DFFD [14] since it has capacity for non-linear deformations as opposed to generally applied linear interpolation which can give smooth result for the surface The process of modification of a generic model fitting feature points detected from given two orthogonal pictures are as follows

- 1 We define control points on a generic model which are corresponding to feature points detected from two views of a person and 27 points for a box surrounding

- 2 We apply global transformations (translation, and scaling) to bring detected feature points to generic model's 3D space We compare two eye extremities in a generic model and a specific person for scaling We check the center of rightmost leftmost un-

most, down-most, front-most, and back-most points of the head for translation.

3. We apply the DFFD on the points of the generic head. The displacement of non-feature points depends on the distance between control points. Since DFFD applies Voronoi and Delaunay triangulation, some points outside triangles of control points are not

modified, the out-box of 27 points can be adjusted locally.

4. Eyes and teeth are recovered to the original shape since modifications may create unexpected deformation for them. Here translation is used after comparison of a generic and individualized heads.

Our system provides a feedback modification of a head between feature detection and a resulted head.

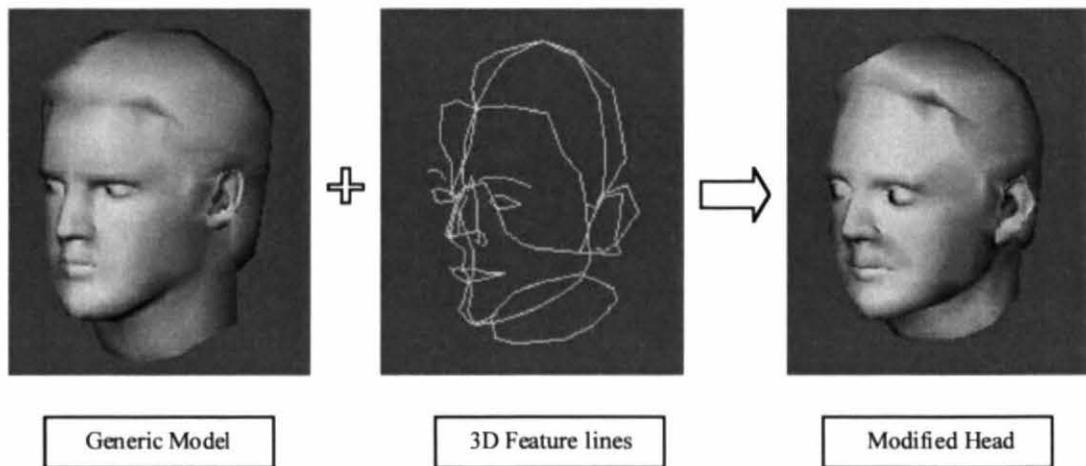


Figure 3-2 A result of DFFD modification comparing with the original head.

3.4 Automatic Texture Mapping

To increase realism, we utilize texture mapping. Texture mapping needs a texture image and coordinate for each point on a head. Since our input is two pictures, texture image generation is needed.

3.4.1 Texture Image Generation

For smooth texture mapping, we assemble two images from front and side views to be one.

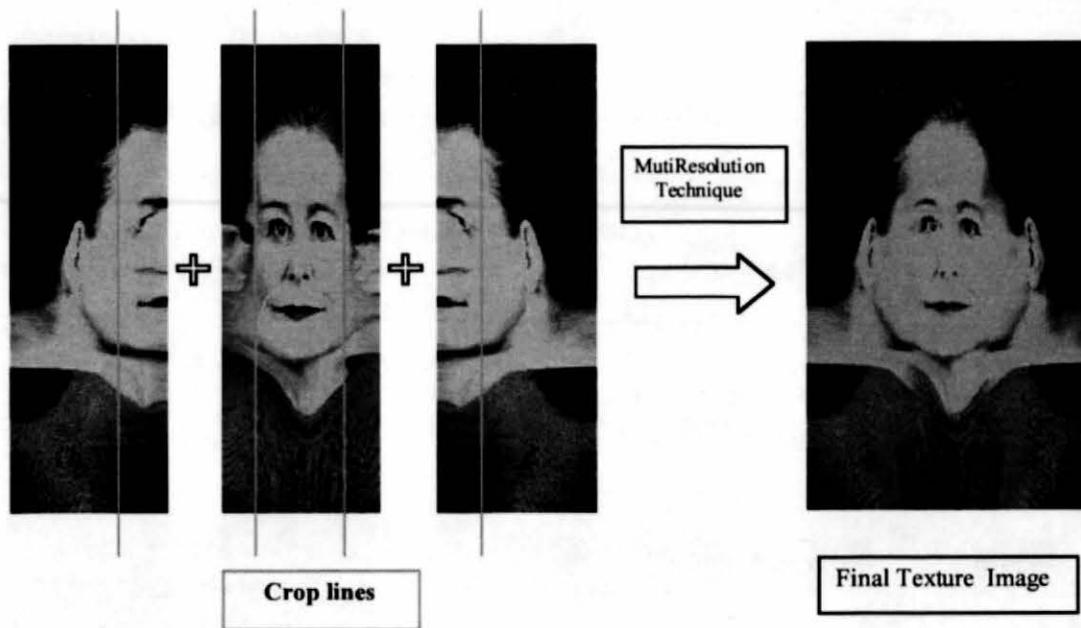


Figure 3-3 Texture Generation from front and side view images. Front and right view after cylindrical projection covering 180° for each. We crop the front and side views around eye ends (shown through white lines) and combine with the left view (flipped from the right view). The last image shows image mosaic of three images, front, right and left using multiresolution spline method.

1. boundaries of two pictures are detected using boundary color information. Since the hair shape is simple in a generic model, the boundary of a side view is modified automatically using information of back head profile feature points detected. It is useful to have nice texture for back part of a neck.
2. The cylindrical projection of each image is done. So the front view will cover from -90° to 90° , right view from 0° to 180° and left one from -180° to 0° .
3. Cropping of two images are processed. Since the front view is good only for a certain range and so is the other. To use conventional blending for wide range using angle variation [9], it is easy to blur certain shape of features. Images for certain points (we use eye extremes because eyes are important to keep high resolution) are cropped on front and side views. Since we have information

about eye positions, it can be done automatically to crop a front view, then we crop right and left views to make the final assembled image be 360° .

4. Image mosaic is applied to produce one image for texture mapping. Since it is almost impossible to take two orthogonal pictures in exactly same condition, just to assemble several images makes the boundaries visible. We use a multiresolution spline method to assemble two images [7].

The cylindrical projections of front and right views and the image mosaic using a multiresolution spline are shown in Figure 3-3.

3.4.2 Texture Fitting

Texture fitting with a composed image is employed to 3D modified head. The main idea for the texture fitting is to map a 2D image to a 3D shape. Texture coordinates of feature points are calculated using detected position

data and a function and cropping lines applied for texture image generation. The problem for texture fitting is how to decide texture coordinates of all points on a surface of a head. The process is as follows.

1. Cylindrical projection to 2D is applied for all points on a surface.
2. Extra points are added to make a convex hull containing all points. Extra points are designed to have texture coordinates on the texture image.
3. The Voronoi triangulation on control (feature) points and extra points is processed.
4. Local Barycentric coordinates of every point with a surrounding Voronoi triangle is calculated.
5. Texture coordinates of each point on a 2D-texture image is obtained using texture coordinates of control points and extra points and corresponding Barycentric coordinate.

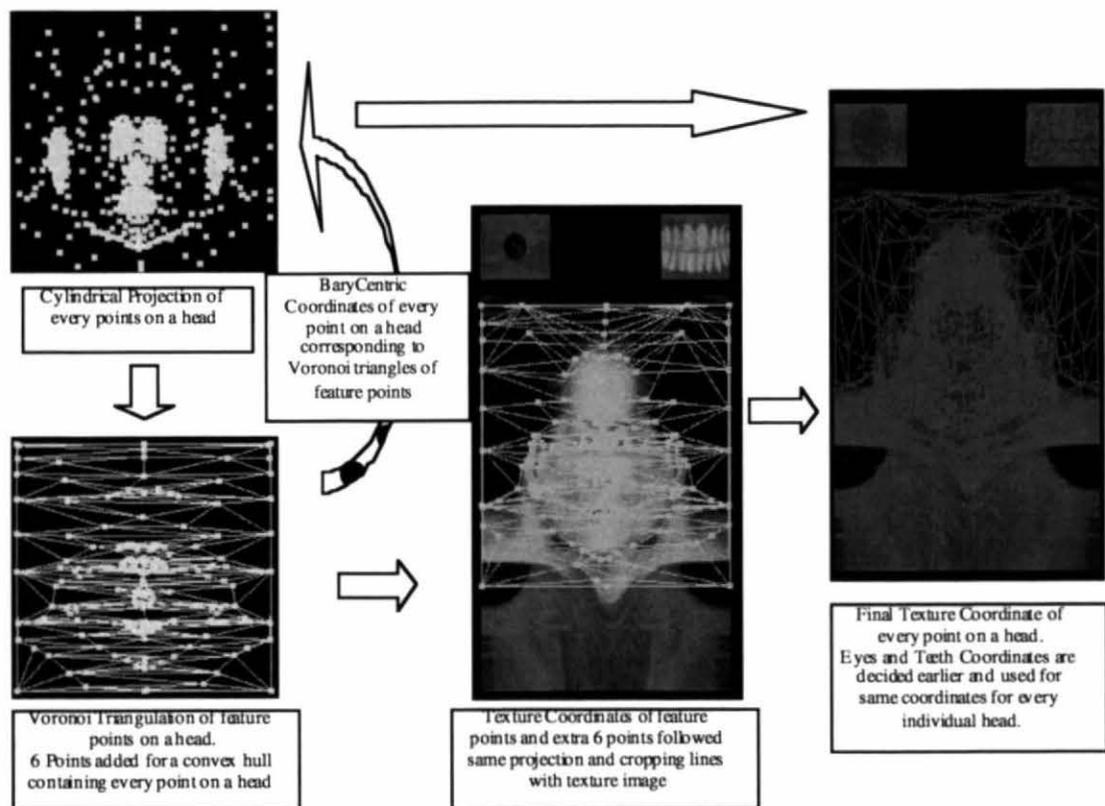


Figure 3-4 Texture fitting process.

3.5 Result and Comparison

A final textured head is shown in Figure 3-6 with input in Figure 3-5, whose process from feature detection to texture mapping takes a few minutes. Compare the output with the

head in Figure 2-1 produced by a designer using *Sculptor* and *TextureFit* which takes weeks of efforts. It has a smoother texture image since it is improved by manual correction by a designer.

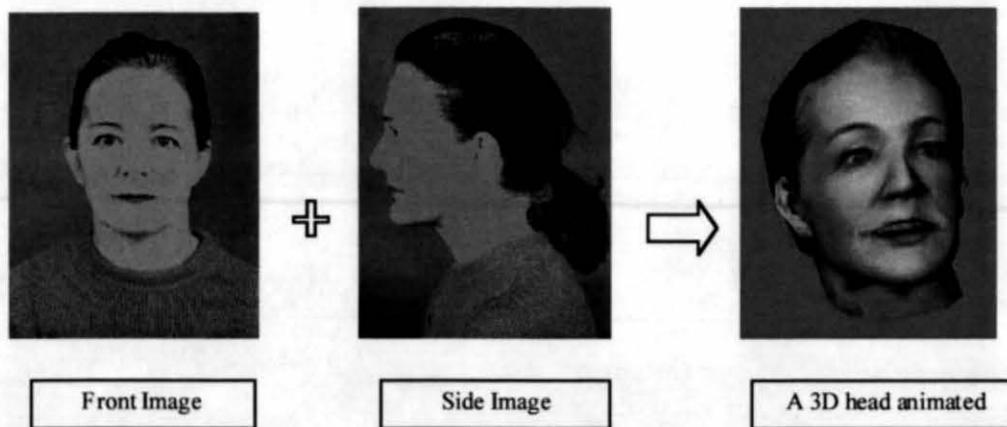


Figure 3-5 Input pictures and final animated head



Figure 3-6 A final reconstructed head. A backside has proper texture too.



Figure 3-7 Face reconstruction by a designer with three pictures (front, diagonal, and side views) using *Sculptor* and *TextureFit* mentioned in Section 1. A texture image part is composed using *PhotoShop*.

4 Range Data

The approach based on 3D digitization to get a range data often requires special purpose high-cost hardware. However when we want to get really highly matched face, it is necessary to

try to have a range data. These data provide with a large number of points usually and it does not have any structure for animation. It is the same as a paper in topological view.

4 1 How to get Range Data?

We describe the main three methods to get range data

Laser Scanning In range image vision system some sensors, such as scanners yield range images. For each pixel of the image, the range to the visible surface of the objects in the scene is known. Therefore spatial location is determined for a large number of points on this surface. An example of commercial 3D digitizer based on laser light scanning, is Cyberware Color Digitizer™ [17]. Lee et al [13] digitized facial geometry through the use of scanning range sensors. However, the approach based on 3D digitization requires special high-cost hardware and a powerful workstation.

Stripe Generator As an example of structured light camera range digitizer, a light striper with a camera and stripe pattern generator can be used for face reconstruction with relatively cheap equipment compared to laser scanners. Stripe pattern is projected on the 3D object surface and it is taken by a camera. With information of positions of projector and camera and stripe pattern a 3D shape can be calculated. Proesmans et al [16] shows a good dynamic 3D shape using a slide projector, by a frame-by-frame reconstruction of a video.

Stereoscopy A distance measurement method such as stereo can establish the correspondence at certain characteristic points. The method uses the geometric relation over stereo images to recover the surface depth. C3D 2020 capture system by the Turing Institute produces many VRML models using stereoscopy method [3].

Most methods have great problem to get hair shape because of the high reflection structure.

4 2 How to give structure?

To get a structured shape for animation, most typical way is to modify an available generic model with structural information such that eyes, lips, nose, hair and so on. Our input is any VRML format with texture image point coordinate data and texture coordinate data. For our experiment we utilized data copied from the Turing Institute [4] where stereoscopic camera is used to generate range data of faces. It has only front face image. We apply similar method with the one in Section 3 with an adaptation method utilizing available detailed shape data. We utilize two step adaptation method, which has rough matching and then detailed matching later. See Figure 4-1 for overall process.

More detailed process follows

- 1 Feature detection for front face picture is processed. Whenever a feature point is detected, the corresponding depth, z coordinate is obtained automatically calculating its neighboring points of given x and y position. Since most cases, a feature point does not correspond to any point in the range data, we collect certain number of nearest points in terms of x and y and then we calculate reverse distance function ratio to get the depth.
- 2 Some points need to be corrected interactively since the hair part is not detected well in the range data requirement. Also back head shape needs to be imagined.
- 3 DFFD is applied with obtained 3D coordinates of given feature points. The result is a rough matching.
- 4 Feature points which are gained from range data are collected using Voronoi triangulation and Barycentric coordinate calculation. The Voronoi triangles and detected

points on a surface are shown in Figure 4-2.

5. Various projections of points in right picture in Figure 4-2 are used with relation to a normal vector of

a corresponding triangle. Once again the nearest points are calculated and reverse distance function is applied to get the corresponding accurate coordinate in a range data.

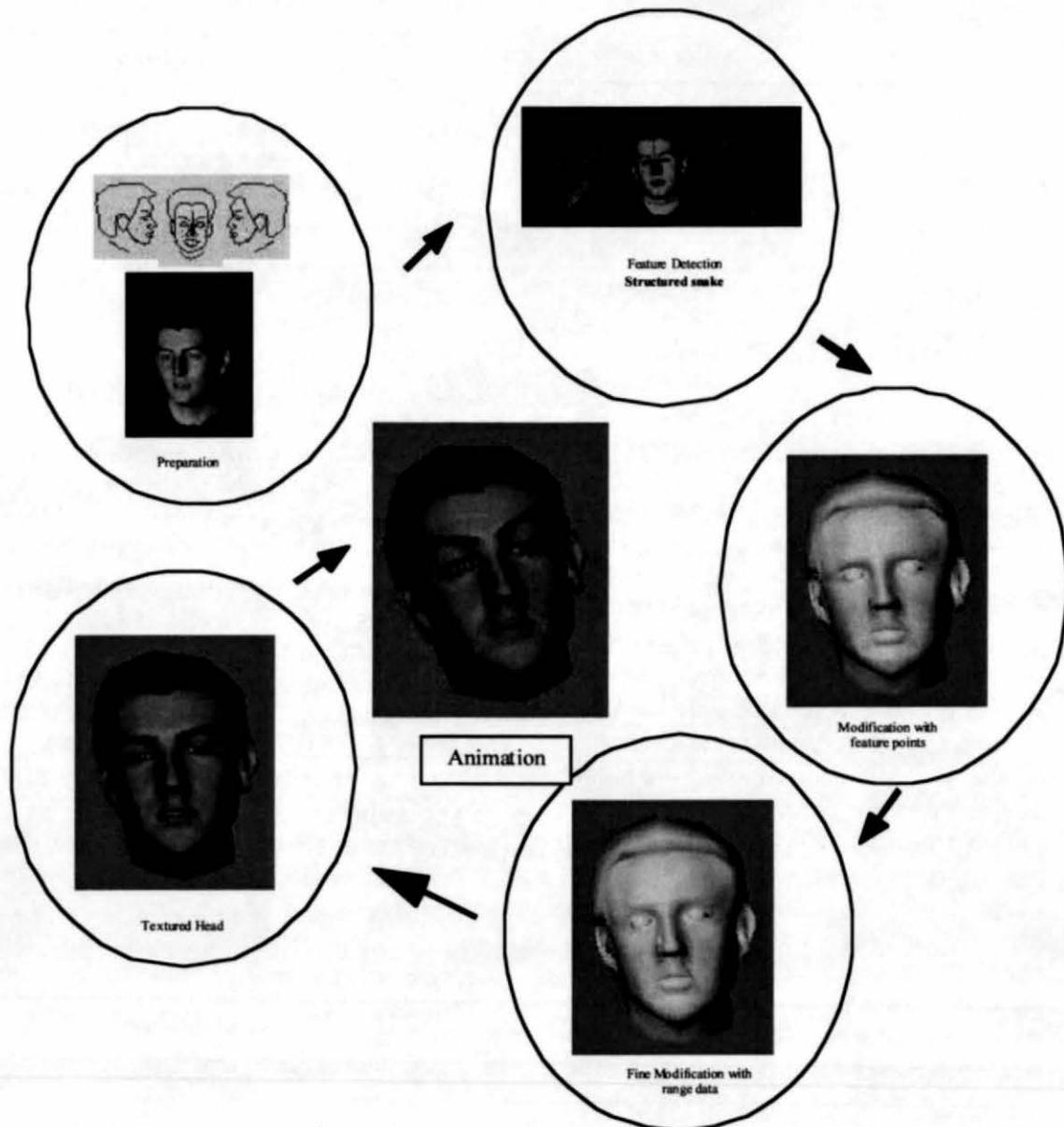


Figure 4-1 The overall process for giving animation structure for range data. The input range data is copied from the Turing Institute. This process can be applied to any range data in VRML format.

Without the first modification step with feature points, it is difficult to

apply projection of points to get detailed matching.

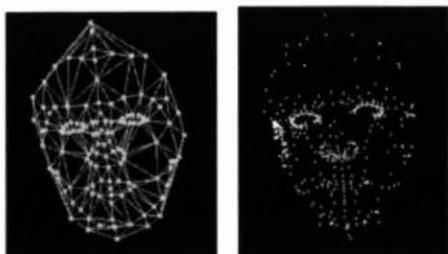


Figure 4-2 Left picture shows Voronoi triangles of feature points which are used for fine modification and the right points which fall inside region for fine modification.

5 Facial Animation System

We employ an approach for deformation and animation of a face, based on pseudo muscle design. The (generic) face model is an irregular structure defined as a polygonal mesh. The face is decomposed into regions where muscular activity is simulated using Rational Free Form Deformations [10]. As model fitting transforms the generic face without changing the underlying structure, the resulting new face can be animated. To simulate the effects of muscle actions on the skin of virtual human face, we define regions on the mesh corresponding to the anatomical descriptions of the regions where a muscle is desired. For example, regions are defined for eyebrows, cheeks, mouth, jaw, eyes, etc. A control lattice is then defined on the region of interest. Muscle actions to stretch, expand, and compress the inside geometry of face are simulated

by displacing or changing the weight of the control points. This deformation model for simulating muscle is simple and easy to perform, natural and intuitive to apply and efficient to use for real time applications.

Facial Motion Control Specification and animation of facial animation muscle actions may be tedious task. There is a definite need for higher level specification which would avoid setting up the parameters involved for muscular actions when producing an animation sequence. The Facial Action Coding System (FACS) [10] has been used extensively to provide a higher level specification when generating facial expressions, particularly in nonverbal communication context. In our multi-level approach as shown in Figure 5-1, motion parameters as Minimum Perceptible Action (MPA) has a corresponding set of visible features such as movement of eyebrows, jaw, or mouth and others occurring as a result of muscle contractions and pulls. The MPAs are used for defining both the facial expressions and the visemes. There are 65 MPAs used in our system which allow us to construct practically any expression and viseme. At the highest level, animation is controlled by a script containing speech and emotions with their duration. Depending on the type of application and input different levels of animation control can be utilized.

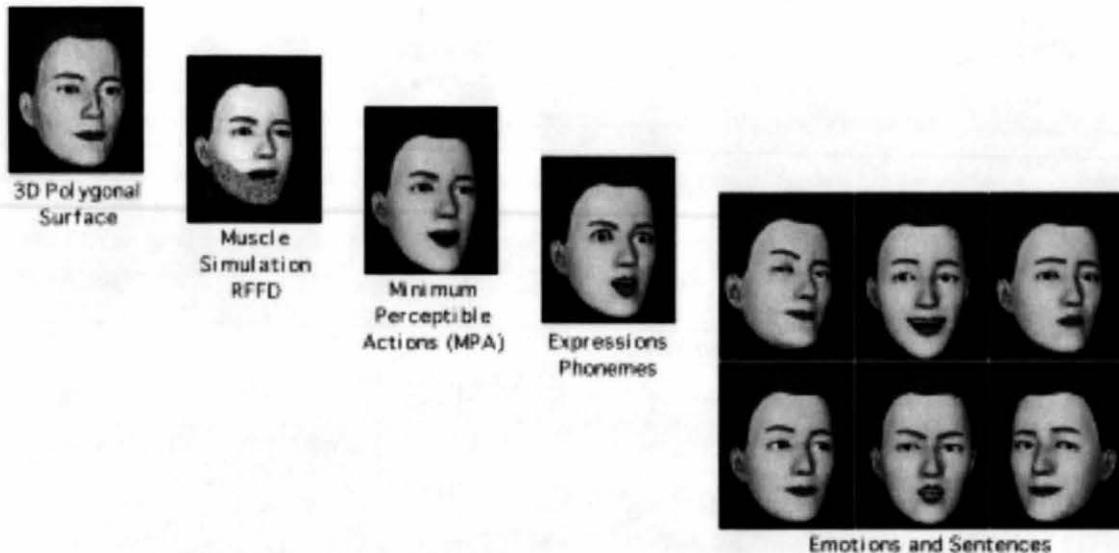


Figure 5-1 Different levels of facial motion control.

6 Conclusion

We described our experience to create a virtual animated face from a real face and compare their inputs and results. First designer oriented reconstruction has a space for artistic sense, but it is time consuming. The second reconstruction method modifying a generic model using two orthogonal pictures needs commercial equipment and takes just few minutes. The third method using range data has the best visual result for output and time, but it requires usually either an expensive or sophisticated equipment.

To have a counter part of a real face into virtual world has a lot of potential in the fields from entertainment to medical application, such as 3D morphing, simulation of generating population, face to face communication through network, and face surgery simulation.

The integration of our reconstruction method of a 3D head with facial feature tracking from video sequence is our ongoing research topic.

7 Reference

- [1] Meet Geri: The New Face of Animation, *Computer Graphics World*, Volume 21, Number 2, February 1998.
- [2] Sannier G., Magnenat Thalmann N., "A User-Friendly Texture-Fitting Methodology for Virtual Humans", *Computer Graphics International'97*, 1997.
- [3] Exhibition On the 10th and 11th September 1996 at the Industrial Exhibition of the British Machine Vision Conference.
- [4] <http://www.turing.gla.ac.uk/turing/copyrigh.htm>
- [5] Takaaki Akimoto, Yasuhito Suenaga, and Richard S. Wallace, Automatic Creation of 3D Facial Models, *IEEE Computer Graphics & Applications*, Sep., 1993.
- [6] P. Beylot, P. Gingsins, P. Kalra, N. Magnenat Thalmann, W. Maurel, D. Thalmann, and F. Fasel, 3D Interactive Topological Modeling using Visible Human Dataset. *Computer Graphics Forum*, 15(3):33-44, 1996.

- [7] Peter J Burt and Edward H Andelson A Multiresolution Spline With Application to Image Mosaics, *ACM Transactions on Graphics*, 2(4) 217-236, Oct, 1983
- [8] Marc Escher, N Magnenat-Thalmann, Automatic 3D Cloning and Real Time Animation of a Human Face *Proc Computer Animation*, IEEE Computer Society, pp 58-66, 1997
- [9] Horace H S Ip, Lijin Yin, Constructing a 3D individual head model from two orthogonal views *The Visual Computer* Springer Verlag, 12 254-266 1996
- [10] Kalra P, Mangili A Magnenat-Thalmann N, Thalmann D, Simulation of Muscle Actions using Rational Free Form Deformations *Proc Eurographics 92 Computer Graphics Forum* Vol 2, No 3, pp 59-69 1992
- [11] M Kass, A Witkin, and D Terzopoulos, Snakes Active Contour Models, *International Journal of Computer Vision* pp 321-331 1988
- [12] Tsuneya Kurihara and Kiyoshi Arai, A Transformation Method for Modeling and Animation of the Human Face from Photographs *Computer Animation*, Springer-Verlag Tokyo, pp 45-58, 1991
- [13] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters, Realistic Modeling for Facial Animation, In *Computer Graphics (Proc SIGGRAPH)*, pp 55-62, 1996
- [14] L Moccozet, N Magnenat-Thalmann, Dirichlet Free-Form Deformations and their Application to Hand Simulation, *Proc Computer Animation*, IEEE Computer Society, pp 93-102 1997
- [15] N Magnenat-Thalmann, D Synthetic Actors in the film *Rendez-vous à Montréal*, *IEEE Computer Graphics and Applications*, 7(12) 9-19 1987
- [16] Marc Proesmans Luc Van Gool Reading between the lines - a method for extracting dynamic 3D with texture In *Proceedings of VRST* pp 95-102, 1997
- [17] http://www.viewpoint.com/free_stuff/cyberscan
- [18] LeBlanc A, Kalra P, Magnenat-Thalmann, N and Thalmann D Sculpting with the 'Ball & Mouse' Metaphor *Proc Graphics Interface '91* Calgary, Canada, pp 152-9, 1991

Facial Deformations for MPEG-4

Marc Escher Igor Pandzic Nadia Magnenat Thalmann

MIRALab CUI
University of Geneva
24 rue du Général Dufour
CH1211 Geneva 4 Switzerland
{Marc Escher Igor Pandzic Nadia Thalmann}@cui.unige.ch
<http://miralabwww.unige.ch/>

Abstract

The new MPEG-4 standard scheduled to become an International Standard in February 1999 will include support not only for natural video and audio but also for synthetic graphics and sounds. In particular representation of human faces and bodies will be supported. In the current draft specification of the standard [MPEG N1901 MPEG N1902] Facial Animation Parameters (FAPs) and Facial Definition Parameters (FDPs) are defined. FAPs are used to control facial animation at extremely low bitrates (approx. 2 kbit/sec). FDPs are used to define the shape of the face by deforming a generic facial model or by supplying a substitute model. We present algorithms to interpret the part of FDPs dealing with the deformation of a generic facial model leading to a personalisation of the model. The implementation starts from a generic model which is deformed in order to fit the input parameters. The input parameters must include the facial feature points and may optionally include texture coordinates and a calibration face model. We apply a cylindrical projection to the generic face in order to interpolate any missing feature points and to fit the texture if supplied. Then we use a Dirichlet Free Form Deformation [Moccozet 97] interpolation method to deform the generic head according to the set of feature points. If the calibration face model is present the fitting method is based on cylindrical projections matching and barycentric coordinates to interpolate the non feature points.

Keywords MPEG 4, SNHC, Facial animation, Face modelling

1 Introduction

ISO/IEC JTC1/SC29/WG11 (Moving Pictures Expert Group MPEG) is currently working on the new MPEG 4 standard [Koenen97 MPEG N1901 MPEG N1902] scheduled to become International Standard in February 1999. In a world where audio visual data is increasingly stored, transferred and manipulated digitally, MPEG 4 sets its objectives beyond plain compression. Instead of regarding video as a sequence of frames with fixed

shape and size and with attached audio information, the video scene is regarded as a set of dynamic objects. Thus the background of the scene might be one object, a moving car, another the sound of the engine, the third etc. The objects are spatially and temporally independent and therefore can be stored, transferred and manipulated independently. The composition of the final scene is done at the decoder, potentially allowing great manipulation freedom to the consumer of the data.

Video and audio acquired by recording from the real world is called natural. In addition to the natural objects, synthetic computer generated graphics and sounds are being produced and used in ever increasing quantities. MPEG 4 aims to enable integration of synthetic objects within the scene. It will provide support for 3D Graphics, synthetic sound, Text to Speech, as well as synthetic faces and bodies. In this paper we concentrate on the representation of faces in MPEG 4 and in particular the methods to produce personalised faces from generic faces.

The following section provides the introduction to the representation of faces in MPEG 4. We explain how Facial Animation Parameters and Facial Definition Parameters are used to define the shape and animation of faces. In section 3 we present our algorithm for the interpretation of Facial Definition Parameters. In the final sections we present the results and conclusions, as well as the ideas for future work.

2 Faces in MPEG-4

The Face and Body animation Ad Hoc Group (FBA) deals with coding of human faces and bodies, i.e. efficient representation of their shape and movement. This is important for a number of applications ranging from communication, entertainment to ergonomics and medicine. Therefore there exists quite a strong interest for standardisation. The group has defined in detail the parameters for both definition and animation of human faces and bodies. This draft specification is based on proposals from several leading institutions in the field of virtual humans research. It is being updated within the

Definition parameters allow detailed definition of body/face shape size and texture Animation parameters allow to define facial expressions and body postures The parameters are designed to cover all naturally possible expressions and postures as well as exaggerated expressions and motions to some extent (e.g. for cartoon characters) The animation parameters are precisely defined in order to allow accurate implementation on any facial/body model

In the following subsections we present in more detail the Facial Animation Parameters (FAPs) and the Facial Definition Parameters (FDPs)

2.1 Facial Animation Parameter set

The FAPs are based on the study of minimal facial actions and are closely related to muscle actions They represent a complete set of basic facial actions and therefore allow the representation of most natural facial expressions The lips are particularly well defined and it is possible to precisely define the inner and outer lip contour Exaggerated values permit actions that are normally not possible for humans but could be desirable for cartoon like characters

All the parameters involving translational movement are expressed in terms of the Facial Animation Parameter Units (FAPU) These units are defined in order to allow interpretation of the FAPs on any facial model in a consistent way producing reasonable results in terms of expression and speech pronunciation They correspond to fractions of distances between some key facial features (e.g. eye distance) The fractional units used are chosen to allow enough precision

The parameter set contains two high level parameters The viseme parameter allows to render visemes on the face without the need to express them in terms of other parameters or to enhance the result of other parameters insuring the correct rendering of visemes Similarly the expression parameter allows definition of high level facial expressions

2.2 Facial Definition Parameter set

An MPEG 4 decoder supporting the Facial Animation must have a generic facial model capable of interpreting FAPs This insures that it can reproduce facial expressions and speech pronunciation When it is desired to modify the shape and appearance of the face and make it look like a particular person/character FDPs are necessary

The FDPs are used to personalise the generic face model to a particular face The FDPs are normally transmitted once per session followed by a stream of compressed FAPs However if the decoder does not receive the FDPs the use of FAPUs insures that it can still interpret the FAP stream This insures minimal operation in broadcast or teleconferencing applications

The Facial Definition Parameter set can contain the following

- 3D Feature Points
- Texture Coordinates for Feature Points (optional)
- Face Scene Graph (optional)
- Face Animation Table (FAT) (optional)

The Feature Points are characteristic points on the face allowing to locate salient facial features They are illustrated in

Figure 1 Feature Points must always be supplied while the rest of the parameters are optional

The Texture Coordinates can be supplied for each Feature Point

The Face Scene Graph is a 3D polygon model of a face including potentially multiple surfaces and textures as well as material properties

The Face Animation Table (FAT) contains information that defines how the face will be animated by specifying the movement of vertices in the Face Scene Graph with respect to each FAP as a piecewise linear function We do not deal with FAT in this paper

The Feature Points Texture Coordinates and Face Scene Graph can be used in four ways

- If only Feature Points are supplied they are used on their own to deform the generic face model
- If Texture Coordinates are supplied they are used to map the texture image from the Face Scene Graph on the face deformed by Feature Points Obviously in this case the Face Scene Graph must contain exactly one texture image and this is the only information used from the Face Scene Graph
- If Feature Points and Face Scene Graph are supplied and the Face Scene Graph contains a non textured face the facial model in the Face Scene Graph is used as a Calibration Model All vertices of the generic model must be aligned to the surface(s) of the Calibration Model
- If Feature Points and Face Scene Graph are supplied and the Face Scene Graph contains a textured face the facial model in the Face Scene Graph is used as a Calibration Model All vertices of the generic model must be aligned to the surface(s) of the Calibration Model In addition the texture from the Calibration Model is mapped on the deformed generic model

In the following section we describe how these options are supported in our system

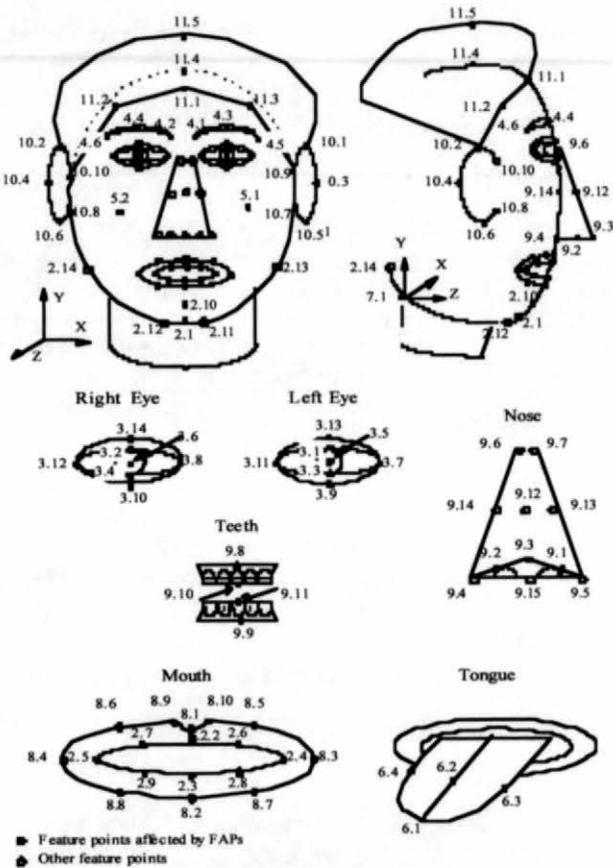


Figure 1: FDP feature point set

3. Algorithms for interpretation of FDPs

3.1 Interpretation of Feature Points only

The first step before computing any deformation is to define a generic head that can be deformed efficiently to any humanoid head by moving specific feature points. The model we use is a 3D polygonal mesh composed of approx. 1500 vertices on which we have fixed a set of vertices that correspond to the feature points defined in MPEG. (Figure2).

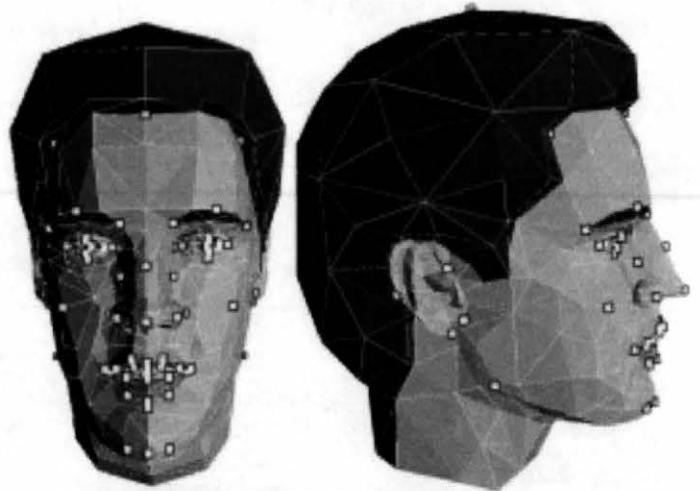


Figure 2: Generic model and feature points

The deformation (fitting) of the generic head is computed using a Dirichlet Free Form Deformation method, which allow a volume deformation using control points while keeping the surface continuity. This method has been developed in MIRALab [Moccozet 97] and uses a Dirichlet diagram to compute the Sibson's local coordinates for the non-feature points interpolation. Figure 3 shows a deformation of the chin on the generic model by dragging the four control points of the chin (dark boxes). Light boxes represent control points.

3.1.1 Missing feature point interpolation

As the Sibson's coordinates calculation is a heavy computation process, it is performed only once for each generic head and saved as a data file. This restrains the use of the DFFD method only to the case when all feature points are available, which may not always be the case. Therefore we perform a pre-processing to interpolate the missing feature points. A cylindrical projection of all the feature points of the generic face, and a Delaunay triangulation of the encoded points are computed. Barycentric coordinates are then calculated for the non-given feature points. Each feature point that had no 3D FDP coordinate at the encoder has now 3 values corresponding each one to the weight of a bounding feature point vertex.



Figure 3: DFFD example

The FDP interpolated coordinate is:

$$X_f = X_i + W_a * (X_{fa} - X_{ia}) + W_b * (X_{fb} - X_{ib}) + W_c * (X_{fc} - X_{ic})$$

Where:

X_f = final 3D coordinate of the non encoded feature point

X_i = initial 3D coordinate of the non encoded feature point

$W_{a,b,c}$ = barycentric coordinate

$X_{fa,b,c}$ = final 3D coordinate of the 3 bounding vertices

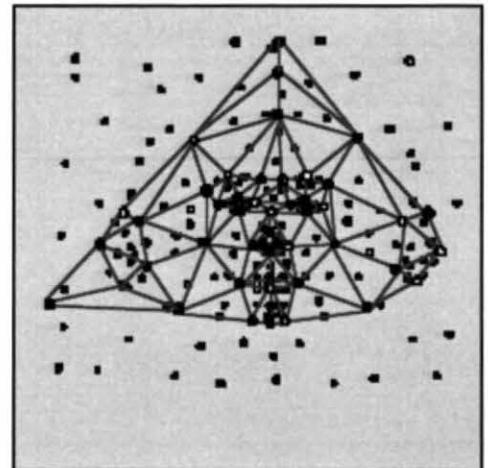
$X_{ia,b,c}$ = initial 3D coordinate of the 3 bounding vertices

Once the 3D position of all the feature points are known we apply the DFFD method to fit the generic head to the extracted/interpolated FDP points.

3.2 Interpretation of Feature Points and Texture

The method we use for computing the texture coordinates uses a cylindrical projection of all the points of the generic 3D face instead of a planar projection. The use of cylindrical projection allows all the points of the head to be texture mapped. Even if generally only one front picture is given as a texture image and only the front part of the face is textured, it is always better to have a more general method that allows a complete mapping of the head.

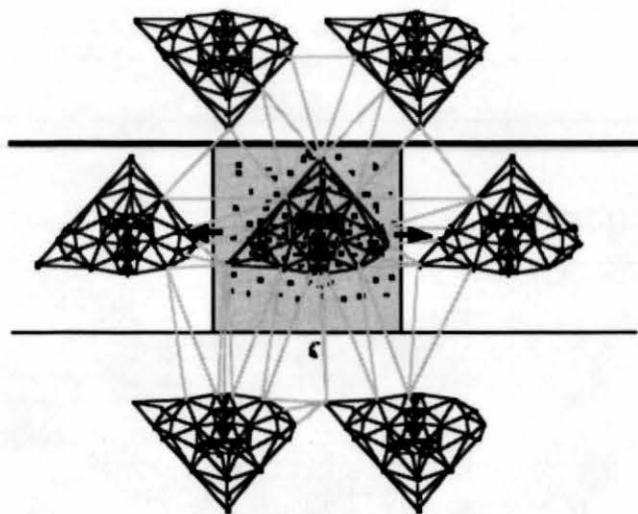
The problem with the cylindrical projection is that the Delaunay triangulation of the projected feature points doesn't include all the non-feature points. (Figure 4.)



- Linked dots: Projected feature points
- Unlinked dots: Projected non-feature points
- Green lines: Feature points triangulation

Figure 4. Cylindrical projection of the head points

This problem can be resolved if we use the property of continuity of the cylindrical projection and some neighbourhood approximation to generate a convex Delaunay triangulation. We use these properties to develop an method that include all the non-feature points in a triangulation of feature points (Figure 5)



- Linked dots: Projected feature points
- Unlinked dots: Projected non-feature points
- Dark lines: Feature points triangulation
- Light lines: Expanded triangulation

Figure 5: Expansion of the feature points

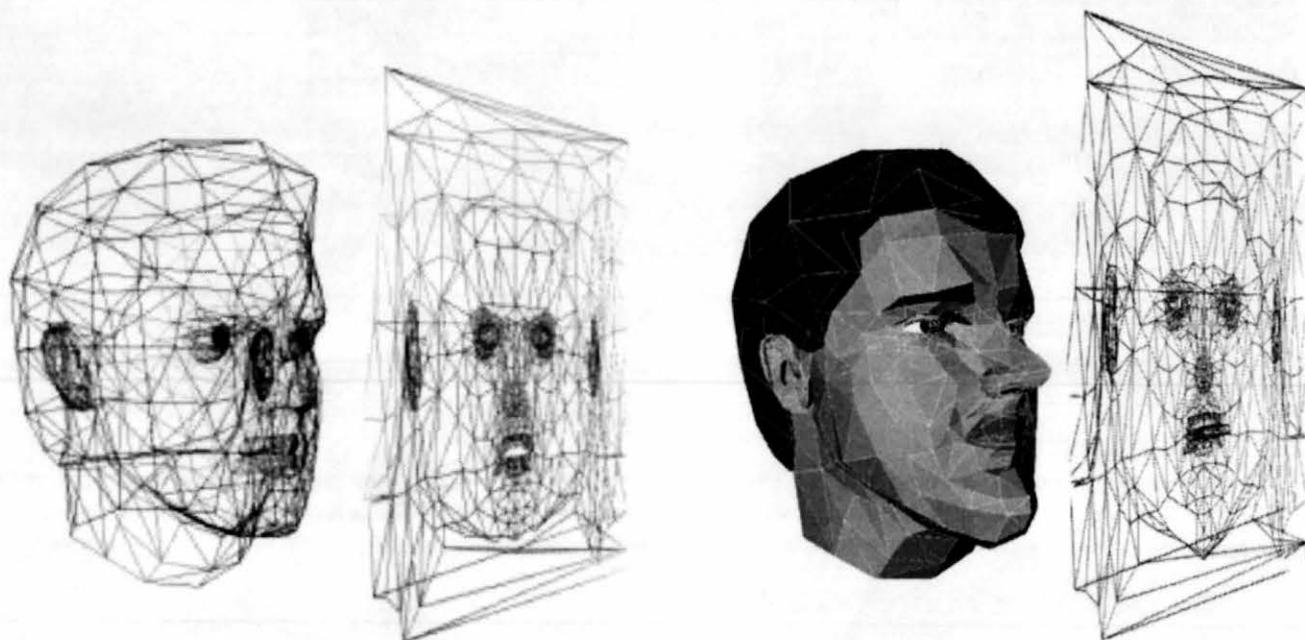
Basically the feature points are duplicated on the left and right side by a horizontal shift. The upper and lower parts are filled with 2 duplications each, using a horizontal symmetry. An "expanded" Delaunay triangulation is then performed, it now includes all the

non-feature points. This method which approximate a spherical projection gives visually acceptable results. (Figure 5)

3.3 Interpretation of Feature Points and Calibration Model

In this profile, a 3D-calibration mesh is given along with the position of its control points. The goal is to fit the generic mesh on the calibration one. Our method starts with the cylindrical projection of both 3D meshes (Figure 6).

The next step is to map the projection of the generic map on the projection of the calibration one. The procedure is exactly the same as the one previously described for the texture fitting, with the use of the 3D projected feature points except of the 2D texture feature points. When the 2D projection of the generic mesh is fitted on the calibration one, we compute the barycentric coordinates of every non-feature points of the generic head in relation with the triangulation of the calibration mesh. At this stage every point of the generic mesh is either a feature point with a corresponding new 3D location, or a non-feature point with barycentric coordinates. The new 3D position of the non-feature points is interpolated using the formula expressed in 3.1. This method works fine for most of the face surface, but for specific regions with high complexity such as the ears, some distortions may appear.



Generic head cylindrical projection

Calibration model cylindrical projection

Figure 6: Cylindrical projection

- Chen_n.fdp, Chen_n.fdp.text, Chen_n.color

The results are shown in the following pictures:

3.4 Interpretation of Feature Points and texture and Calibration Model

The addition of texture is done in the same way as described in 3.1. I.e. Expansion and triangulation of the texture feature points, local barycentric coordinates extraction for every non-feature points of the 3D-head mesh. Concatenation of a mouth and eye picture on the texture image in order to apply texture on hidden parts. (Figure 7) The iris colour of the eyes is selected automatically from the texture picture by extraction the HLS parameters from the most open eye. The concatenated eye picture is then modified to match these parameters.



Figure 7: Complete texture image

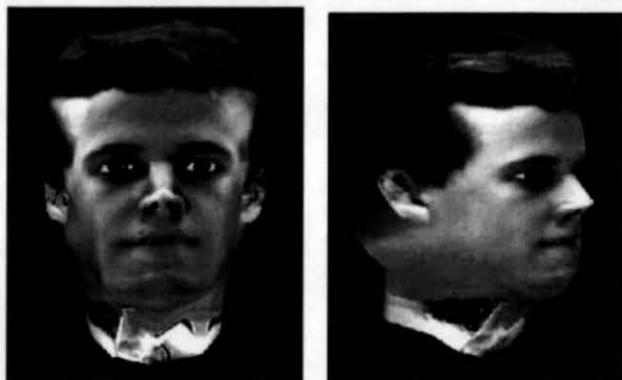


Claude_n.fdp

4. Results

We were involved in the CE FBA3. The experiment was carried out with test FDP and texture files all available from the FBA Core Experiments home page:

- Jim_n.fdp, Jim_n.fdp.text, Jim_n.color
- Claude_n.fdp, Claude_n.fdp.text, Claude_n.color



Jim_n.fdp



Chen_n.fdp

Figure 8: Final results

5. Conclusions

This paper has described some techniques of face fitting and texturing adapted to the actual definitions of the MPEG-4 SNHC Face Definition Parameters. We have presented our implementation of texturing using cylindrical projection and in particular a method for generating an encompassing delaunay triangulation by expanding the projected feature points. The face modelling using 3D feature points or a calibration model, using extensively delaunay triangulation and barycentric coordinates has also been explained. Finally some results have been shown.

6. Acknowledgements

This research is financed by the ACTS project AC057 VIDAS.

7. References

- [Boulic 95] Boulic R., Capin T., Huang Z., Kalra P., Lintermann B., Magnenat-Thalmann N., Moccozet L., Molet T., Pandzic I., Saar K., Schmitt A., Shen J., Thalmann D., "The Humanoid Environment for Interactive Animation of Multiple Deformable Human Characters", *Proceedings of Eurographics '95*, 1995.
- [Kalra 92] Kalra P., Mangili A., Magnenat Thalmann N., Thalmann D., "Simulation of Facial Muscle Actions Based on Rational Free Form Deformations", *Proc. Eurographics '92*, pp.59-69., 1992.
- [Kalra 93] Kalra P. "An Interactive Multimodal Facial Animation System", *PhD Thesis nr. 1183*, EPFL, 1993.
- [Koenen 97] Koenen R., Pereira F., Chiariglione L., "MPEG-4: Context and Objectives", *Image Communication Journal, Special Issue on MPEG-4*, Vol. 9, No. 4, May 1997.
- [Moccozet 97] Moccozet L. Magnenat Thalmann N., "Dirichlet Free-Form Deformation and their Application to Hand Simulation", *Proc. Computer Animation '97, IEEE Computer Society*, pp.93-102.
- [MPEG-N1901] "Text for CD 14496-1 Systems", ISO/IEC JTC1/SC29/WG11 N1886, MPEG97/November 1997.

[MPEG N1902] "Text for CD 14496 2 Video" ISO/IEC
JTC1/SC29/WG11 N1886

Face to Virtual Face

Nadia Magnenat Thalmann, Prem Kalra, Marc Escher

MIRALab, CUI

University of Geneva

24, rue du General-Dufour

1211 Geneva, Switzerland

Email {thalmann, kalra, escher}@cui.unige.ch

URL <http://miralabwww.unige.ch/>

Abstract

The first virtual humans appeared in the early eighties in such films as Dreamflight (1982) and The Juggler (1982). Pioneering work in the ensuing period focused on realistic appearance in the simulation of virtual humans. In the nineties, emphasis has shifted to real-time animation and interaction in virtual worlds. Virtual humans have begun to inhabit virtual worlds and so have we. To prepare our place in the virtual world, we first develop techniques for the automatic representation of a human face, capable of being

animated in real time using both video and audio input. The objective is for one's representative to look, talk and behave like oneself in the virtual world. Furthermore, the virtual inhabitants of this world should be able to see our avatars and to react to what we say and to the emotions we convey.

This paper sketches an overview of the problems related to the analysis and synthesis of face to virtual face communication in a virtual world. We describe different components of our system for real-time interaction and communication between a cloned face representing a real person and an autonomous virtual face. It provides an insight into the various problems and gives particular solutions adopted in reconstructing a virtual clone capable of reproducing the shape and movements of the real person's face. It includes the analysis of the facial expression and speech of the cloned face, which can be used to elicit a response from the autonomous virtual human with both verbal and non-verbal facial movements synchronised with the audio voice.

We believe that such a system can be exploited in many applications such as natural and intelligent human-machine interfaces, virtual collaboration work, virtual learning and teaching, and so on.

Keywords Virtual Human, Virtual Face, Clone, Facial Deformation, Real Time Facial Animation, Autonomous Virtual Human, Virtual Dialog, Phoneme Extraction, 3D Feature Points

1 Introduction

When thinking of virtual humans it is natural to cast them in real films. Who would not be intrigued to see Adjani opposite Bogart in the romantic thriller of the season? However, the reasons behind the increasingly complex techniques for portraying and directing virtual humans almost real enough to fool their mothers are as much economic and educational as artistic. Virtual humans in the large majority of applications, will be used to explore situations that have not yet happened. Couldn't we avoid a great deal of disappointment by seeing how we would look before having our hair styled, or cut differently? More dramatically wouldn't it relieve a lot of anxiety if candidates for plastic surgery could see their smiles before undergoing a face lift or chin reconstruction? Wouldn't it be better to learn the anatomy and physiology of the human body or about problems in speech pathology by watching and manipulating synthetic actors? And why persist in using robots in ergonomics if we can simulate real people?

The first computerized human models were created 20 years ago by airplane and car manufacturers. The main idea was to simulate a very simple articulated structure for studying problems of ergonomics.

In the seventies researchers developed methods to animate human skeletons mainly based on interpolation techniques. Bodies were represented by very primitive surfaces like cylinders, ellipsoids or spheres. At the same time, the first experimental facial animation sequences appear¹

The Juggler (1982) from Information International Inc², was the first realistic human character in computer animation, the results were very impressive, however, the human shape was completely digitized, the body motion had been recorded using 3D rotoscopy and there was no facial animation. The first 3D procedural model of human animation was used in producing the 12 minutes film DREAMFLIGHT³, one of the first to feature a 3D virtual human.

In the eighties, researchers started to base animation on the key frame animation, parametric animation, and late eighties on the laws of physics. Dynamic simulation made it possible to generate complex motions with a great deal of realism. However, an ordinary human activity like walking is too complex to be simulated by the laws of dynamics alone. Two people, with the same physical characteristics, do not move in the same way. Even one individual does not move in the same way all the time. A behavioral approach to human animation is necessary in the near future to lend credibility to such simulations.

But the main complexity in the animation of virtual humans is the problem of integrating many techniques. True virtual humans should be able to walk, talk, grasp objects, show emotions and communicate with their environment. This will be the next challenge.

The face is a relatively small part of a virtual human, but it plays an essential role in communication. We look at faces for clues to emotions or even to read lips. It is a particular challenge to simulate these aspects. The ultimate objective therefore is to model human facial anatomy exactly, including its movements with respect both to structural and functional aspects. Recent developments in facial animation include physically-based approximation to facial tissue and the reconstruction of muscle contractions from video sequences of human facial expressions. Problems of correlation between emotions and

voice intonation have been also studied. Ensuring synchronization of eye motion, facial expression of emotion and the word flow of a sentence, as well as synchronization among several virtual humans is at the heart of our new facial animation system at the University of Geneva.

The capability of animating virtual humans through a task-level language requires a deep understanding of how tasks should be specified. The process of interpreting natural language instructions involves subtle and sometimes unexpected connections between language and behavior. When the behavior is to be portrayed by a virtual human, various questions are raised regarding the types and roles of planning, reasoning, constraint satisfaction, human capabilities, and human motion strategies.

In the context of interactive animation systems, the relationship between the user as animator and the virtual human as synthetic actor should be emphasized. With the availability of graphics workstations able to display complex scenes rendering about a million polygons per second, and with the advent of such interactive devices as the SpaceBall, EyePhone, and DataGlove, it is possible to create computer-generated characters based on a full 3D interaction metaphor in which the specifications of deformations or motion are given in real-time. True interaction between the animator and the actor requires two-way communication: not only can the animator give commands to the actor but the actor must also be able to respond behaviorally and verbally. Finally, we may aspire to a virtual reality where virtual humans participate fully: real dialog between the animator and the actor.

This paper is an account of a face-to-face virtual interaction system where a clone representing a real person, can dialog with another virtual human who is autonomous, in a

virtual world. The dialog consists of both verbal and other expressive aspects of facial communication between the two participants. Section 2 gives an overview of the problem and describes major contributions related to the different aspects. Section 3 concentrates on our system and describes different components of the system. Section 4 presents issues related to the standardization of parameters for defining the shape and animation of the face. Future trends are outlined in the concluding remarks.

2 Problem domain and related work

To clone is to copy. In our context, cloning means reproducing a virtual model of a real person in the virtual world. Here, our interest is restricted to one component of human figure: the face. The face is the most communicative part of a human figure. Even a passive face conveys a large amount of information, and when it comes to life and begins to move, the range of motions it offers is remarkable: we observe the lips, teeth, and tongue for speech, eyes and head movements for additional elements of dialog, and flexing muscles and wrinkle lines for emotions.

Developing a facial clone model requires a framework for describing geometric shapes and animation capabilities. Attributes such as surface color and textures must also be taken into account. Static models are inadequate for our purposes: the model must allow for animation. The way facial geometry is modeled is motivated largely by its animation potential.

Even though most faces have similar structure and the same set of features there is considerable variation from one individual face to another. These subtle and small differences make the individual face recognizable. In modeling the face of a real person, these aspects have to be captured for the model to be identifiable as a clone.

Prerequisites for cloning a face are analyses of several aspects necessary for its reconstruction: its shape and its movements due to both emotions and speech. This requires techniques from various fields. Shape includes geometrical form as well as other visual characteristics such as color and texture. Input for shape reconstruction may be drawn from photographs and/or scanned data. The synthesis of facial motion involves deforming its geometry over time according to physical or ad hoc rules for generating movements conveying facial expressions and speech. The input for the facial motion of the clone will be the facial expressions and/or the speech of the real person.

In the rest of this section, we review work related to shape reconstruction, synthesis of facial animation, the analysis and tracking of facial motion, and facial communication. The vast domain of speech analysis and recognition is beyond the scope of this review.

2.1 3D Shape reconstruction

Geometrical representation

Among the variety of ways of representing a face geometrically, the choice should be one that allows for precise shape, effective animation and efficient rendering.

Two broad categories may be distinguished volume representation and surface representation⁴ Volume representation may be based on constructive solid geometry (CSG) primitives or volume elements (voxels) from medical images However, volume representation has not been widely adopted for facial animation because CSG primitives are too simple to produce reasonable face shapes Voxels are high resolution data need to be segmented from a huge voxel map and require data thinning Largely for these reasons, the animation using volumic data is computationally intensive

Surface primitives and structures are currently the preferred geometrical representations for faces Among surface description techniques are polygonal surfaces parametric surfaces and implicit surfaces In a polygonal surface representation a face is a collection of polygons regularly or irregularly shaped The majority of existing models use polygonal surfaces primarily because of their simplicity and the hardware display facilities available for polygons on most platforms The parametric surfaces use bivariate parametric functions to define surfaces in three dimensions, e.g. bicubic B-spline surfaces^{5 6} The advantage of these models is that they have smooth surfaces and are determined using only a few control points However, local high-density details for eyes and mouth are difficult to add Hierarchical B-splines developed by Forsey and Bartels⁷ enable more local detail without the need to add complete rows or columns of control points Wang⁸ has used the hierarchical B-splines for modeling and animating faces An implicit surface is an analytic surface defined by a scalar field function⁹ Interaction with implicit surfaces is difficult with currently available techniques, and these have not yet been used for facial modeling

Facial features

A face consists of many parts and details. Researchers tend to focus on the visible external skin surface of the face from neck to the forehead -- 'the mask' -- for facial animation study and research. However, it is necessary to add features like eyeballs, teeth, tongue, ears, hair, etc. to obtain realistic results. In addition, these features are essential to the recognition of particular individuals.

Facial data acquisition

Face models rely on data from various sources for shapes, color, texture, etc. In constructing geometrical descriptions, two types of input should be distinguished: three-dimensional and two-dimensional.

Three-dimensional input

Use of a 3D digitizer/scanner would seem to be the most direct method for acquiring the shape of a face. A 3D digitizer involves moving a sensor or locating device to each surface point to be measured. Normally, the digitized points are the polygonal vertices (these can also be the control points of parametric surfaces). There are several types of 3D digitizers employing different measurement techniques (mechanical, acoustic, electromagnetic). Polhemus, an electromagnetic digitizer, has been used by many researchers for modeling faces^{10, 11}. In other cases, a plaster model has been used for marking the points and connectivities. This procedure is not automatic and is very time-consuming.

Laser-based scanners, such as Cyberware, can provide both the range and reflectance map of the 3D data in a few seconds. The range data produce a large regular mesh of points in a

cylindrical coordinate system. The reflectance map gives color and texture information. One of the problems with this method is the high density data provided. Another is that the surface data from laser scanners tend to be noisy and have missing points. Some postprocessing operations are required before the data can be used. These may include relaxation membrane interpolation for filling in the missing data¹², filter methods, e.g. hysteresis blur filters, for smoothing data¹³, and adaptive polygon meshes to reduce the size of the data set for the final face model¹⁴.

As an alternative to measuring facial surfaces, models may be created using interactive methods like sculpturing^{15, 16}. Here the face is designed and modeled by direct and interactive manipulation of vertex positions or surface control points. This, however, presupposes design skills and sufficient time to build the model. When constructing a clone, relying on subjective visual impressions may not be accurate or rapid enough.

Two dimensional input

There are a number of methods for inferring 3D shape from 2D images. Photogrammetry of a set of images (generally two) can be used for estimating 3D shape information. Typically, the same set of surface points are located and measured in at least two different photographs. This set of points may even be marked on the face before the pair of photographs is taken. The measurement can be done manually or using a 2D digitizer. A better method takes account of perspective distortion by using a projection transformation matrix determined by six reference points with known 3D coordinates¹⁷.

Another approach is to modify a canonical or generic face model to fit the specific facial model using information from photographs of the specific face^{18, 19, 20}. This relies on the

fact that humans share common structures and are similar in shape. The advantages here are that no specialized hardware is needed and that the modified heads all share the same topology and structure and hence can be easily animated.

2.2 Animation techniques

Since the clones we are interested in creating will not remain static but will have to move like real people, we briefly review work done in animating synthetic models of the face. The different approaches employed for animation are primarily dictated by the context of application. There are five basic kinds of facial animation: interpolation, parametrization, pseudo-muscle based, muscle-based, and performance-driven.

Interpolation resembles the key frame approach used in conventional animation in that the desired facial expression is specified for a particular point in time, defining the keyframe. An in-betweening algorithm computes the intermediate frames. This approach, which has been used to make several films^{21, 22}, is very labor intensive and data-intensive.

Parametric animation models make use of local region interpolation, geometric transformations, and mapping techniques to manipulate the features of the face²³. These transformations are grouped together to create a set of parameters. Sets of parameters can apply to both the conformation and the animation of the face.

In pseudo-muscle based models, muscle actions are simulated by abstract notions of muscles, where deformation operators define muscle activities²⁴. The dynamics of different

facial tissues is not considered. The idea here is not to simulate detailed facial anatomy exactly but to design a model with a few parameters that emulate muscle actions.

There are no facial animation models yet, based on complete and detailed anatomy. Models have, however, been proposed and developed using simplified structures for bone, muscle, fatty tissue and skin. These models enable facial animation through particular characteristics of the facial muscles. Platt and Badler²⁵ used a mass-spring model to simulate muscles. Waters²⁶ developed operators to simulate linear and sphincter muscles having directional properties. A physically-based model has been developed where muscle actions are modeled by simulating the tri-layer structure of muscle, fatty tissues and the skin²⁷. Most of these methods do not have real-time performance.

Performance-based animation uses data from real human actions to drive the virtual character. The input data may come from interactive input devices such as a DataGlove, instrumented body suits, video or laser-based motion-tracking systems. The next section elaborates methods of tracking facial motion from video-based input.

2.3 Facial motion tracking

The ability to track facial motion accurately promises unique opportunities for new paradigms of human-computer interaction. The task of automatically and faithfully tracking and recognizing facial dynamics, however, without invasive markers or special lighting conditions, remains an active exploratory area of research.

One of the simplest approaches is to track markers placed directly on the face of the person whose facial motion we wish to track. With retroreflective markers located on the performer's face, the problem of tracking becomes one of tracing a set of bright spots on a dark field. The technique is effective when there are discrete markers and they remain in view at all times.

Tracking motion robustly is difficult. There have been some efforts in computer vision to find reasonable solutions. Both model-based and image-based methods have been employed. One approach makes use of a deformable template, assuming that each face consists of features like eyes, nose, and mouth that are uniquely identifiable²⁸. A potential energy function for the image, formulated as combination of terms due to valleys, edges, peaks, image and internal potential, is minimized as a function of the parameters of the template. The limitation of this method is that it becomes unstable in the absence of a feature as the template continues to look for it.

Snakes, or active contours, have also been used to track features²⁹. A snake is basically the numerical solution of a first order dynamical system. The human face has many feature lines and boundaries that a snake can track. Snakes can also be used for estimating muscle contraction parameters from a sequence of facial images³⁰. The image intensity function is transformed into a planar force field by computing the magnitude of the gradient of the image intensity. Some methods use combination of image processing techniques for different features of the face as snakes may not be stable for all the regions³¹. B-splines have also been used for defining snakes where the control points are time varying³². Methods using snakes are generally not very robust as the numerical integration may become unstable.

Optic flow computation has also been used to recover image motion parameters corresponding to facial expressions³³ Optic flow can be estimated by tracking the motion of pixels from frame to frame This enables us to capture the dynamics of facial expressions and hence add a temporal aspect to the characterization of a facial expression, something beyond the scope of the Facial Action Coding System³⁴ In another approach, optical flow is used in conjunction with deformable models to estimate shape and motion in human faces³⁵ Methods using optical flow require high textural detail in the images as the computations rely on pixel level detail

2.4 Facial communication

Facial communication among virtual humans has recently attracted much attention Cassell et al³⁶ describe a system which generates automatic speech and gestures, including facial expressions for modeling conversation among multiple human-like agents This system, however, does not include cloning aspects Thorisson³⁷ presents a mechanism for action control using a layered structure, for communicative humanoids The virtual environment consists of Gandalf a simplified caricatural face used as the autonomous actor

There can be four different situations for face to virtual face communication in a virtual world These are real face to virtual face, cloned face to cloned face, virtual face to virtual face, and cloned face to virtual face The four cases are briefly described as follows

Real Face to Virtual Face

Here a real face communicates with a virtual face who is autonomous and has some intelligence to understand what the real face conveys. This may have two types of input from the real face: video input from the camera and speech from audio (microphone). The emotions of the real person are recognized by facial features extraction and tracking from the video input and the audio is converted to the text and phonemes. These are used by the autonomous virtual face to respond in a believable manner through his/her facial expressions and speech (see Figure 1). Mimicking the input from the real face video or speech are the special cases of this situation.

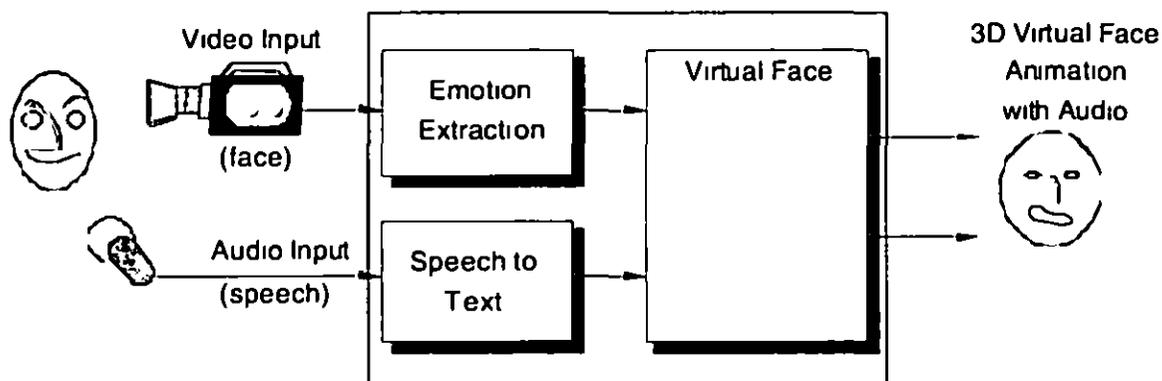


Figure 1 Real face to virtual face

Cloned Face to Cloned Face

In a networked 3D virtual environment communication may exist between the participants located at different sites represented by their 3D clones. This requires construction of 3D

clone and facial motion capture of the real face from the video camera input for each participant. The extracted motion parameters are mapped to the corresponding facial animation parameters for animating the cloned models. The audio or speech input is also reproduced on each clone. Figure 2 shows the modules for such a communication.

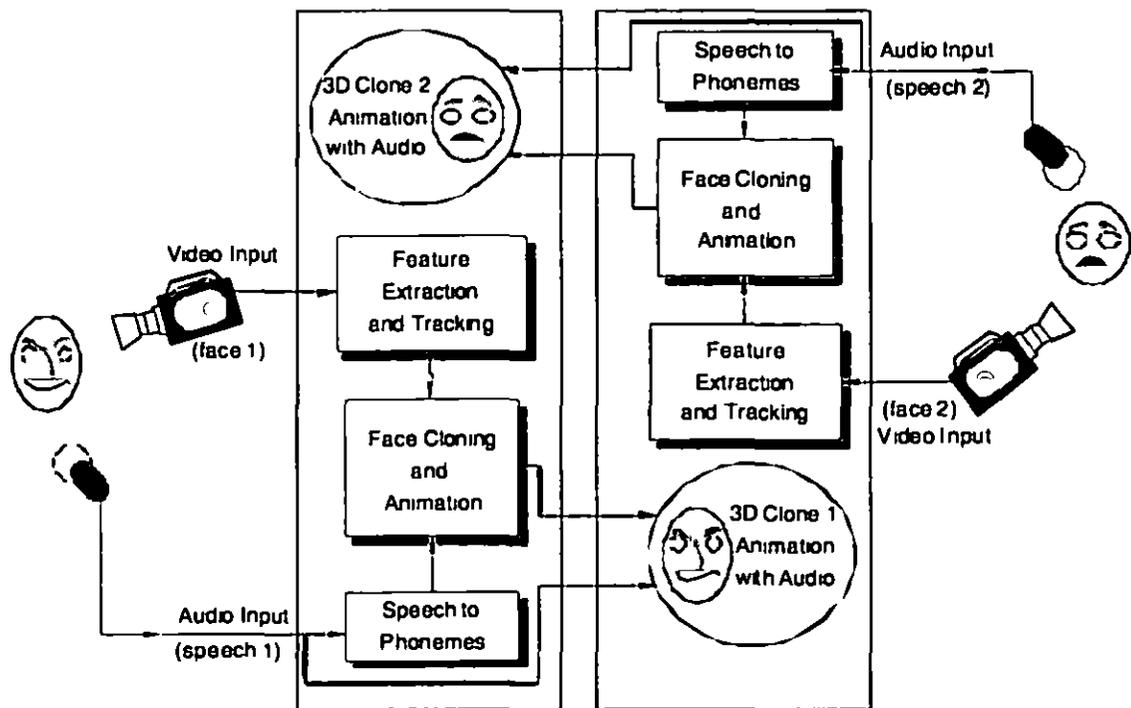


Figure 2 Cloned face to cloned face

Virtual Face to Virtual Face

A virtual face who is autonomous may communicate with another autonomous virtual face. The communication may involve both speech and facial emotions (Figure 3). This situation does not require any cloning aspects. The autonomy of the virtual humans gives them the

intelligence and capacity to understand what is conveyed by the other virtual human and generate a credible response

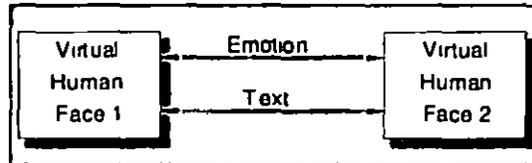


Figure 3 Virtual face to virtual face

Cloned Face to Virtual Face

A virtual environment inhabited by the clones representing real people and virtual autonomous human would require communication between a cloned face and virtual face. This needs the cloning and mimicking aspects to reconstruct the 3D model and movements of the real face. The autonomous virtual face is able to respond and interact through facial expressions and speech. This is of particular interest when there is more than one real participant being represented by their clone with one or more autonomous virtual humans. Although the participant may not be interested in watching their clone, they would be able to see the presence of the other real participants. Figure 4 shows the configuration of the communication between a cloned face and a virtual face.

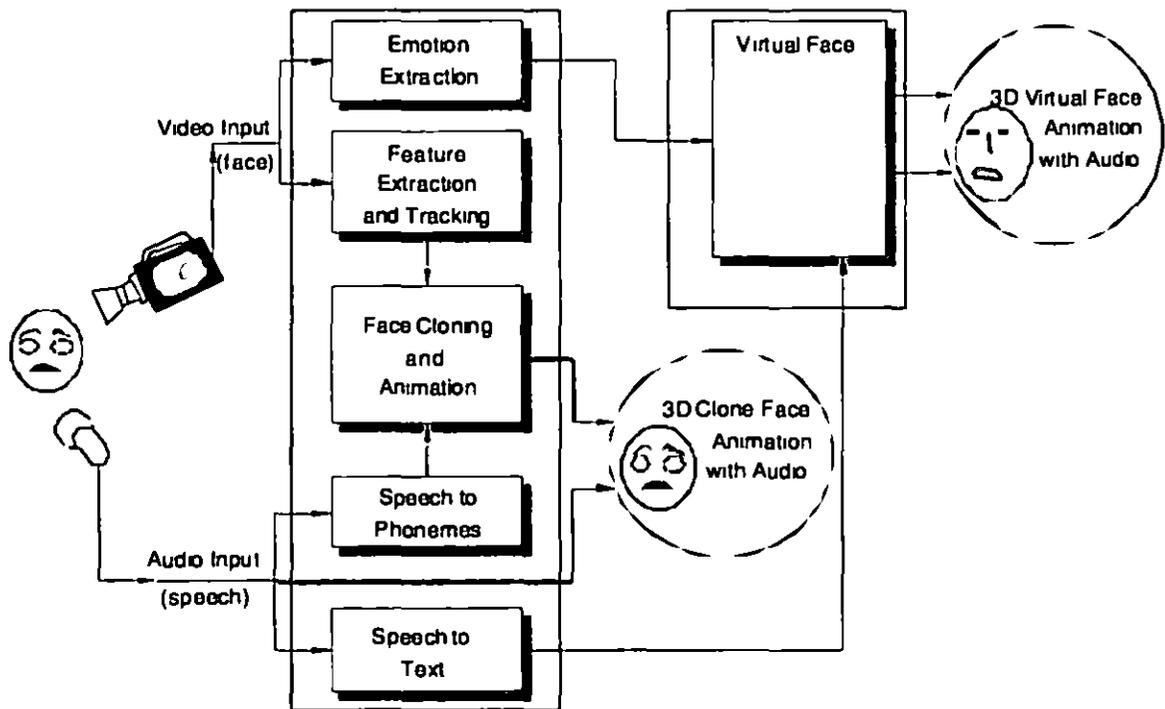


Figure 4 Cloned face (of real face) to virtual face

The communication between a cloned face and virtual face addresses all the problems involved for facial cloning and communication and is considered as a general case for all the above four situations

We now describe our system, with its different modules that enables face to virtual face communication in a virtual environment inhabited by the virtual clone of a real person and the autonomous virtual human. The system concentrates only on the face. Other body parts, though often communicative in real life, are for the moment considered passive. Although the system is described considering one cloned face and one virtual autonomous face, it can have multi cloned and multi virtual faces.

3 A description of our system

The general purpose of the system is to be able to generate a clone of a real person and make it talk and behave like the real person in a virtual world. The virtual world may contain another virtual human, who is autonomous. dialog and other communication can be established between the clone and the autonomous virtual human. The autonomous virtual human understands the expressions and speech of the real person (conveyed via the clone) and communicates using both verbal and non verbal signals. The dialog is simulated in a 3D virtual scene which can be viewed from different remote sites over the network.

Implementing such a system necessitates solving many independent problems in many fields: image processing, audio processing, networking, artificial intelligence, virtual reality and 3D animation. In designing such a system we divide it into modules, each module being a logical separate unit of the entire system. These modules are: 3D face reconstruction, animation, facial expression recognition, audio processing, interpretation and response generation, and audio-visual synchronization.

The following sub-sections describe the different modules of the system.

3.1 3D Face reconstruction

The 3D face model is constructed from a generic/canonical 3D face using two orthogonal photographs, typically a front and a side view. The process is also referred to as model-fitting as it involves transforming the generic face model to the specific face of the real person. First, we prepare two 2D templates containing the feature points from the generic

face – these points characterize the shape and morphology of the face -- for each orthogonal view. If the two views of the person to be represented have different heights and sizes, a process of normalization is required. The 2D templates are then matched to the corresponding features on the input images³⁸. This employs structured discrete snakes to extract the profile and hair outline and filtering methods for the other features like the eyes, nose, chin, etc. The 3D coordinates are computed using a combination of the two sets of 2D coordinates. This provides the set of target positions of the feature points. The generic non feature points are modified using a deformation process called Dirichlet Free Form Deformation (DFFD)³⁹. DFFD is a generalized method for free form deformation (FFD)⁴⁰ that combines traditional FFD with scattered data interpolation methods based on Delaunay/Dirichlet diagrams. DFFD imposes no constraint on the topology of the control lattice. Control points can be specified anywhere in the space. We can then perform model-fitting using a set of feature points defined on the surface of the generic face as the DFFD control points⁴¹. For realistic rendering we use texture mapping to reproduce the small details of facial features which may not show up in the gross model fitting. Figure 5 shows the different steps for constructing a 3D face model for the clone.

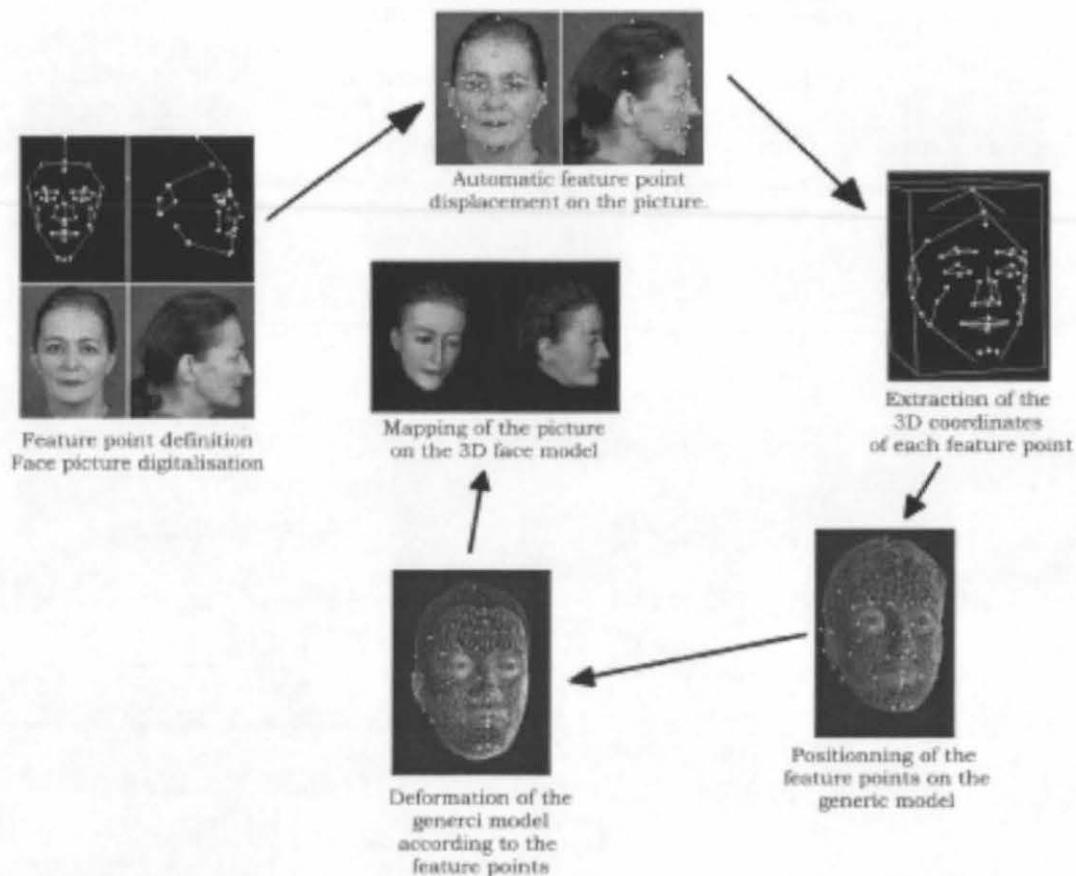


Figure 5: Steps for clone construction.

3.2 Animation

A face model is an irregular structure defined as a polygonal mesh. The face is decomposed into regions where muscular activity is simulated using rational free form deformations ⁴². As model fitting transforms the generic face without changing the underlying structure, the resulting new face can be animated. Animation can be controlled on several levels. On the lowest level we use a set of 65 minimal perceptible actions (MPAs) related to the muscle movements. Each MPA is a basic building block for a facial motion parameter that controls a visible movement of a facial feature (such as raising an eyebrow or closing the eyes). This set of 65 MPAs allows construction of practically any expression and phoneme. On a

higher level, phonemes and facial expressions are used, and at the highest level, animation is controlled by a script containing speech and emotions with their duration and synchronization. Depending on the type of application and input, different levels of animation control can be utilized. Figure 6 shows these levels.

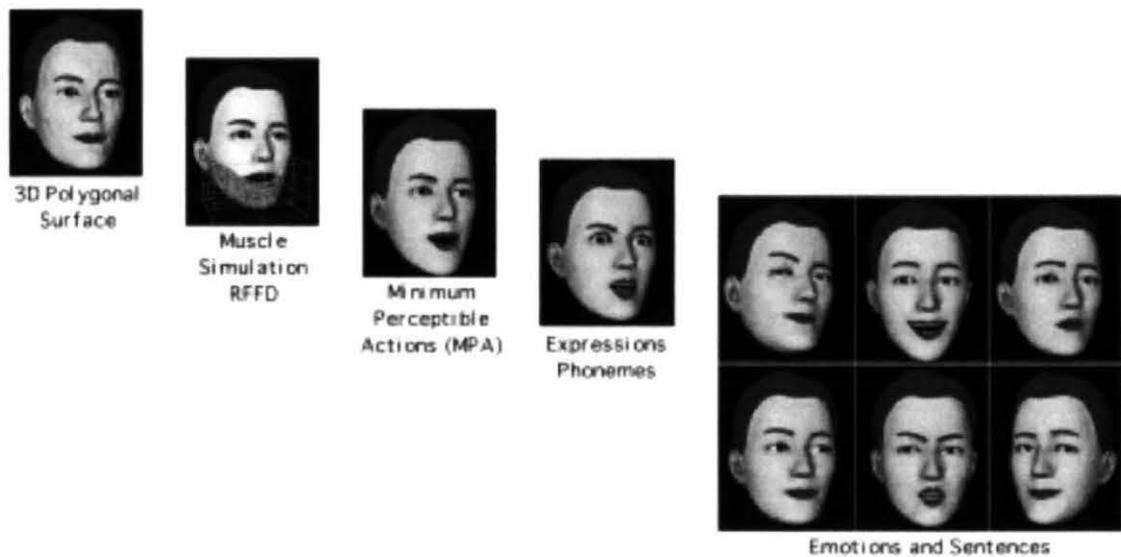


Figure 6: Different levels of animation control.

3.3 Facial expression recognition

Accurate recognition of facial expression from a sequence of images is complex. The difficulty is greatly increased when the task is to be done in real time. In trying to recognize facial expression with a reasonable degree of accuracy and reliability in real time, a few simplifications are inevitable. We focus our attention on only a few facial features for detection and tracking. The method relies on a “soft” mask which is a set of points defined

on the frontal face image. During the initialization step, the mask can be interactively adjusted to the image of the real person, permitting detailed measurements of facial features⁴³. Other information such as a color sample of the skin, background and hair, etc. are also stored. The feature detection method is based on color sample identification and edge detection. The feature points used for facial expression recognition are concentrated around the mouth, the neck, the eyes, the eyebrows, and the hair outline. The data extracted from previous frame are used only for features which are easy to track (e.g., the neck edges), thus avoiding the accumulation of error. In order to reproduce the corresponding movements on the virtual face a mapping is carried out from the tracked features to the appropriate MPAs, the basic motion parameters for facial animation. This allows us to mimic the facial expression of the real person on their clone. Figure 7 illustrates how the virtual face of the clone is animated using input from the real person's facial expressions.



(a): Working session with the user in front of a CCD camera.



(b): Real time facial expression recognition and animation of the clone's face.

Figure 7: Facial animation of the clone's face.

All this information still does not tell us enough about the mood and/or the emotion of the real person. The person's mood and emotions are important information to be input to the process of formulating the autonomous virtual human's response. We employ a rule-based approach to infer the emotion of the person. These rules are simple mapping of active MPAs to a given emotion, however, these are limited to only a few types of emotion. Another approach to recognizing the basic emotion is to use a neural network with the automatically extracted facial features as input associated to each emotion. We are currently making some experiments for classifying the basic emotions (surprise, disgust, happiness, fear, anger, and sadness) using neural network approach where the input is the extracted data from video and the output is one of the six emotions. Difficulty remains in identifying

a blend of basic emotions. This may be partially resolved by identifying the dominant features of the basic emotion depicted and masking the others.

3.4 Audio processing

The processing of the audio signal (which contains most of the dialog content) to text is non-trivial. Recently, some results have been reported in speaker-dependent, restrained contextual vocabulary for continuous speech (90% success rate)⁴⁴. There are, however, many techniques for speech recognition. Typically, they involve the following processes: digital sampling of speech, acoustic signal processing (generally using spectral analysis), recognition of phonemes, groups of phonemes, and then words (hidden Markov modeling systems are currently the most popular, basic syntactic knowledge of the language may aid the recognition process). However, these techniques are time-consuming and not appropriate for real-time applications. Recent developments suggest that recognition is possible in real time, when used in a simplified context^{45, 46}. Audio analysis is essential if the semantics of the input speech is required before formulating a response.

Text-to-speech or speech synthesis is another important part of audio processing. The usual way is to split textual speech into small units, generally phonemes, the smallest meaningful linguistic unit. Each phoneme has a corresponding audio signal. It is no easy matter, however, to combine them to produce a fluent speech. One commonly employed solution is to use diphones instead of just phonemes, which contain the transitions between pairs of phonemes. This squares the number of the elements to be processed in the database but improves the quality considerably. Inflections corresponding to punctuation are added to generate a more human-like voice.

To animate the face using audio input, the audio phonemes are mapped to their corresponding visual output, called the viseme. Since the viseme is defined as set of MPAs, as previously mentioned, it can be applied to any face. Figure 8 shows the same viseme on two different face models.

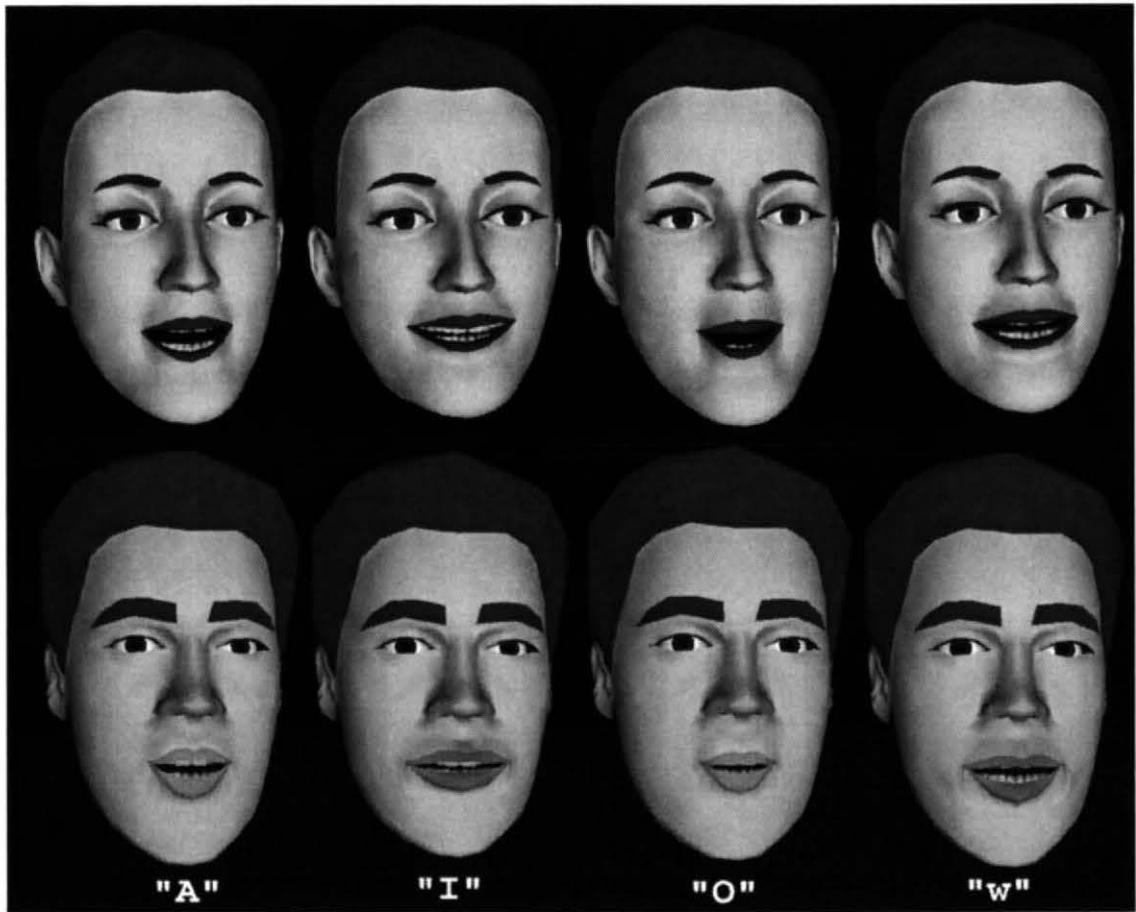


Figure 8: Animation using audio input.

3.5 Interpretation and response generation

For there to be virtual dialog between the clone and an autonomous virtual human, the autonomous virtual human should be able to “understand” the speech and the emotions of

the clone. This requires the addition of an intelligent module to act as the 'brain' of the autonomous participant. The information it has to process is composed of the text of the clone's speech and its emotions inferred from facial expressions. The analysis may involve techniques of natural language processing (see Figure 9)

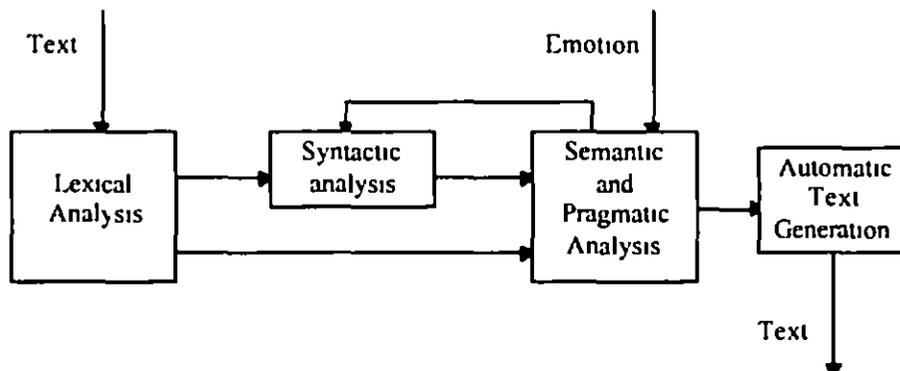


Figure 9 Natural language processing steps

For each word, the lexical analysis retrieves information stored in a lexicon and then the syntactic analysis relies on a set of grammatical rules to parse each sentence in order to determine the relations among the various groups of words. The semantic analysis infers the meaning of the sentence and also generates the output used in the automatic response generation. This step can be bolstered by pragmatic analysis of real-world states and knowledge exemplified in our case by the emotions conveyed by the clone.

Our simplified prototype is an automaton where the final state after the syntactic and semantic analysis is one of the responses in a database available to the autonomous virtual human. This database contains a number of pre-defined utterances, each associated with an emotional state. The current system has limited intelligence, however, this is being extended and elaborated including complex knowledge analysis and treatment.

3.6 Audio visual synchronisation

To produce bimodal output, where the virtual human exhibits a variety of facial expressions while speaking, audio and visual output must be synchronized. To synchronize the sound with the animation, the sound stream is stored in an audio buffer, and the animation, in terms of MPAs, is stacked in an MPA buffer. An MPA synchronizer controls the trigger to both buffers. At present for text to phoneme we are using Festival Speech Synthesis System from University of Edinburgh, UK, and for phoneme to speech (synthetic voice) we use MBROLA from Faculté Polytechnique de Mons, Belgium. Both are public domain software.

3.7 Network issues

From the networking point of view, the virtual dialog system acts like a simple client-server whose architecture is that of an ongoing networked collaborative virtual environment. Virtual Life NETWORK (VLNET)⁴⁷. The clients include the user, as represented by the participation of their clone, as well as one or more external (non-active) viewers.

To reduce bandwidth, only animation or reconstruction parameters of the clone are transmitted to the server. This means that all the video and sound processing has to be computed by the client. The server returns the parameters of the scene to the clients. These include the deformation of objects in the scene, the parameters for speech synthesis of the autonomous virtual human and the compressed speech of the clients and/or clones. The

client is free to select a point of view of the scene, with the default being determined by the position of the clone. The client has to reconstruct the view according to the parameters given by the server, decompress the audio and synthesize the speech from the phonemes. Figure 10 shows interaction of the clone with another virtual human while playing chess through network when seen by a third viewer.



Figure 10: Interaction between virtual humans through network.

3.8 Complete pipeline

Figure 11 gives the overview of the complete pipeline showing the data flow among the various modules. The input can be in various forms or media, here we are primarily concerned with video and audio inputs. However, this can be extended for other input such as 3D trackers for body gestures and positions.

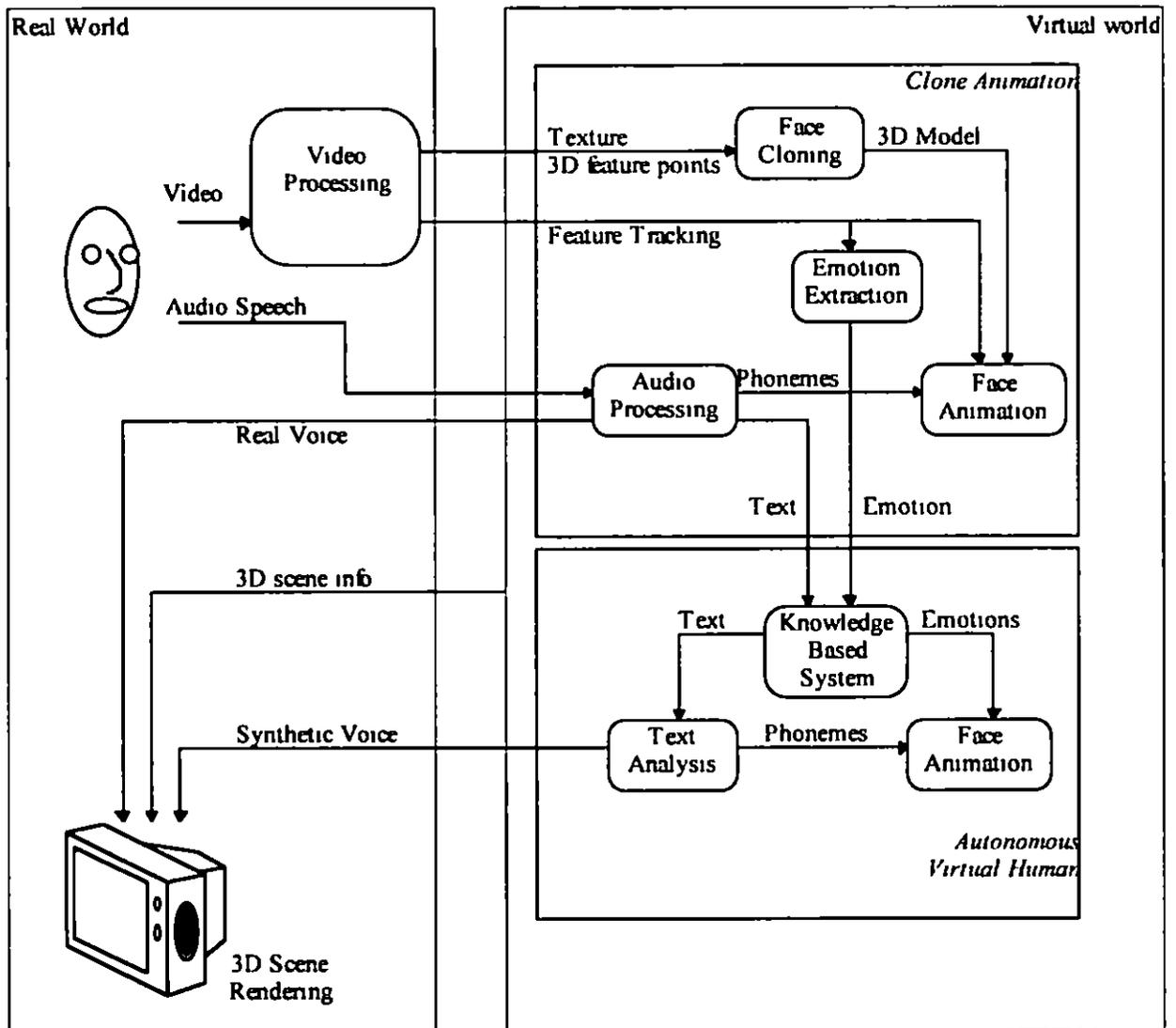


Figure 11 Data flow of the virtual dialogue system

The audio signal is analyzed to extract the speech content (text) and the phonemes composing this text. In order to synchronize with the animation, we need the onset and duration of each phoneme extracted. The data from the video signal is processed to extract all the visual information about the user, including 3D feature points for modeling the clone face and tracking features for animation.

The autonomous virtual human, a separate entity, receives text and emotions as input. This is processed by the knowledge-based module, which then also provides the output response, containing text and emotion. The text is processed to generate temporized phonemes for generating facial deformations as well as synthetic speech. The voice is synchronized with the face deformation, which also accounts for the emotions. Summarizing, the output of the system includes the virtual clone reproducing the motion and speech of the real person, and the autonomous virtual human communicating with the real person as represented by the clone via both speech and emotion-conveying expressions.

Figure 11 shows the interaction between only one real face (via clone) and one virtual face, but it can be extended to multi-clone and multi-virtual face communication. This system proves that in principle it is possible—at least in a limited context, to capture and track people's face shapes and movements, recognize and interpret their facial expressions, and produce a virtual dialog, all in real time.

4 Standardisation for SNHC

SNHC (Synthetic Natural Hybrid Coding) is a subgroup of MPEG-4 that is devising an efficient coding for graphics models and compressed transmission of their animation parameters specific to the model type ⁴⁸ The University of Geneva is making a contribution to the group as a working partner in VIDAS a European project formulating a standard set of parameters for representing the human body and the face For faces, the Facial Definition Parameter set (FDP) and the Facial Animation Parameter set (FAP) are designed to encode facial shape and texture, as well as animation of faces reproducing expressions, emotions and speech pronunciation

FAP is based on the study of minimal facial actions (like the MPAs in our system) and are closely related to muscle actions They represent a complete set of basic facial actions and allow the representation of most natural facial expressions The lips are well defined to take into account inner and outer contours Exaggerated FAP values permit actions that are not normally possible for humans, but could be desirable for cartoon-like characters

All parameters involving motion are expressed in terms of the Facial Animation Parameter Units (FAPU) They correspond to fractions of distances between key facial features (e.g distance between the eyes) The fractional units are chosen to ensure a sufficient degree of precision

The parameter set contains three high-level parameters The viseme parameter allows direct rendering of visemes on the face without the intermediary of other parameters, as well as enhancing the result by applying other parameters, thus insuring the correct rendering of

visemes. The current list of visemes is not intended to be exhaustive. Similarly, the expression parameter allows the definition of high-level facial expressions.

The FDPs are used to customize a given face model to a particular face. They contain 3D feature points (e.g. mouth corners and contours, eye corners, eyebrow ends, etc.), 3D mesh (with texture coordinates if appropriate) (optional), texture image (optional) and other (hair, glasses, age, gender) (optional).

5 Conclusion and future work

Providing the computer with the ability to engage in face-to-face communication - an effortless and effective interaction among real-world people - offers a step toward a new relationship between humans and machines. This paper describes our system, with its different components, which allows real-time interaction and communication between a real person represented by a cloned face and an autonomous virtual face. The system provides an insight into the various problems embodied in reconstructing a virtual clone capable of reproducing the shape and movements of the real person's face. It also includes the syntactic and semantic analysis of the message conveyed by the clone through his/her facial expressions and speech, which can then be used for provoking a credible response from the autonomous virtual human. This communication consists of both verbalizations and non-verbal facial movements synchronized with the audio voice.

We have shown through our system that in a simple situation, it is possible to gather considerable perceptual intelligence by extracting visual and aural information from a face.

This knowledge can be exploited in many applications: natural and intelligent human-machine interfaces, virtual collaborative work, virtual learning and teaching, virtual studios, etc. It is not too far-fetched to imagine a situation, real or contrived, where a fully cloned person (face, body, clothes) interacts in a virtual environment inhabited by other virtual humans, clones or autonomous. However, further research effort is required to realize this in real time with realistic and believable interaction. Figure 12 shows a situation where a virtual clone is hanging out with famous virtual actors in a bar (the image is produced from a non-real-time script-based 3D animation).



Figure 12: A bar scene with virtual humans.

Future work will involve recognition and integration of other body gestures and full body communication among virtual humans. We have done some work in the direction of communication between virtual humans using facial and body gestures⁴⁹. However, cloning of complete virtual humans including animation still needs a lot of research.

Acknowledgements

The research is supported by the Swiss National Foundation for Scientific Research and the European ACTS project VIDEO ASSISTED WITH AUDIO CODING AND REPRESENTATION (VIDAS). The authors would like to thank the members of MIRALab, in particular Laurence Suhner, Laurent Moccozet, Jean-Claude Moussaly, Igor S. Pandžić, Marlène Poizat, and Nabil Sidi-Yagoub.

References

- ¹ Parke FI (1974) *A Parametric Model for Human Faces* Ph D Thesis University of Utah Salt Lake City UT Dec 1974
- ² Information International (1982) *SIGGRAPH Film Festival Video Review 1982* ACM Press
- ³ Magnenat Thalmann N Bergeron P Thalmann D (1982) DREAMFLIGHT a fictional film produced by 3D computer animation Proc Computer Graphics Online Conf pp 353 368 and in the on line film festival (won first award)
- ⁴ Parke FI Waters K (1996) *Computer Facial Animation* A K Peters Wellesley Massachusetts USA
- ⁵ Nahas M Huitric H Sanintourens M (1988) Animation of a B-spline figure *The Visual Computer* 3(5) 272 276 March 1988
- ⁶ Waite CT (1989) *The Facial Action Control Editor FACE A parametric facial expression editor for computer generated animation* Master s Thesis MIT Media Arts and Sciences Cambridge MA Feb 1989
- ⁷ Forsey DR Bartels RH (1988) Hierarchical B-spline refinement *Computer Graphics (Proc SIGGRAPH 88)* 22(4) 205 212 Aug 1988
- ⁸ Wang CL (1993) *Langwidere Hierarchical spline based facial animation system with simulated muscles* Master s Thesis University of Calgary Calgary October 1993

-
- ⁹ Blinn J (1982) A generalization of algebraic surface drawing *ACM Transactions on Graphics* 1(3) 235 256 July 1982
- ¹⁰ Kleiser J (1989) A fast, efficient, accurate way to represent the human face In *State of the Art in Facial Animation SIGGRAPH 89 Tutorial Vol 22* pp 37-40
- ¹¹ Magnenat Thalmann N Thalmann D (1987) *Synthetic actors in computer-generated 3D films* Springer Verlag Tokyo 1987
- ¹² Lee Y Terzopoulos D Waters K (1995) Realistic modeling for facial animation *Computer Graphics (Proc SIGGRAPH 95)* 29(4) 55 62
- ¹³ Williams I (1990) Performance driven facial animation *Computer Graphics (Proc SIGGRAPH 90)* 24(4) 235 242 1990
- ¹⁴ Terzopoulos D Waters K (1991) Techniques for realistic facial modeling and animation In *Magnenat Thalmann N and Thalmann D editors Proc Computer Animation 91* pp 59-74 Springer Verlag Tokyo 1991
- ¹⁵ Elson M (1990) Displacement facial animation techniques *In State of the Art in Facial Animation SIGGRAPH 90 Course Notes No 26* pp 21-42
- ¹⁶ LeBlanc A Kalra P Magnenat Thalmann N Thalmann D (1991) Sculpting with the ball and mouse metaphor *Proc Graphics Interface 91* pp 152 159 Calgary
- ¹⁷ Parke FI editor (1990) State of the Art in Facial Animation *SIGGRAPH 90 Course Notes No 26*

-
- ¹⁸ Akimoto T Suenaga Y (1993) Automatic creation of 3D facial models *IEEE Computer Graphics & Applications* pp 16-22 Sep 1993
- ¹⁹ Horace HS Ip Lijun Yin (1996) Constructing a 3D individualized head model from two orthogonal views *The Visual Computer* Springer-Verlag 12:254-266
- ²⁰ Kunihara T Arai K (1991) A transformation method for modeling and animation of the human face from photographs *Proc Computer Animation 91* eds N Magnenat Thalmann and D Thalmann Springer Verlag, pp 45-58
- ²¹ Kleiser Walczak (1988) *Sextone for President* Short Animated Film 1988 *SIGGRAPH Video Review* ACM
- ²² Kleiser Walczak (1989) *Don't Touch Me* Short Animated Film 1989 *SIGGRAPH Video Review* ACM
- ²³ Parke FI (1982) Parameterized models for facial animation *IEEE Computer Graphics and Applications* 2(9) 61-68 Nov 1982
- ²⁴ Magnenat Thalmann N Primeau NE Thalmann D (1988) Abstract muscle actions procedures for human face animation *The Visual Computer* 3(5) 290-297 1988
- ²⁵ Platt SM Badler NI (1981) *Animating facial expressions* *Computer Graphics (Proc SIGGRAPH 81)* 15(3) 245-252 1981

-
- ²⁶ Waters K (1987) A muscle model for animating three dimensional facial expressions *Computer Graphics (Proc SIGGRAPH 87)* 21(4) 17-24
- ²⁷ Terzopoulos D Waters K (1990) Physically based facial modeling analysis and animation *The Journal of Visualization and Computer Animation* 1(4) 73-80
- ²⁸ Yuile AI Cohen DS Hallinan PW (1989) Feature extraction from faces using deformable templates *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 89)* pp 104-109
- ²⁹ Kass M Witkin A Terzopoulos D (1988) Snakes active contour models *International Journal of Computer Vision* 1(4) 321-331
- ³⁰ Terzopoulos D Waters K (1993) Analysis and synthesis of facial image sequences using physical and anatomical models *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(6) 569-579 1993
- ³¹ Pandzic IS Kalra P Magnenat Thalmann N Thalmann D (1994) Real time facial interaction *Displays* 15 (3) 157-163
- ³² Blake A Isard M (1994) 3D position attitude and shape input using video tracking of hands and lips *Computer Graphics (Proc SIGGRAPH 94)* 28 185-192 July 1994
- ³³ Essa I Pentland A (1995) Facial Expression Recognition using a Dynamic Model and Motion Energy *Proc International Conference on Computer Vision* pp 360-367 IEEE Computer Society

-
- ³⁴ Ekman P Friesen WV (1978) *Manual for the Facial Action Coding System* Consulting Psychology Press Inc Palo Alto CA 1978
- ³⁵ De Carlo Douglas Metaxas D (1996) The integration of optical flow and deformable models with applications to human face shape and motion estimation *CVPR 96* pp 231-238
- ³⁶ Cassell J Pelachaud C Badler NI Steedman M Achorn B Becket T Deouville B Prevost S Stone M Animated Conversation Rule based generation of facial expression, gesture and spoken intonation for multiple conversational agents *Proc SIGGRAPH 94* pp 413-420
- ³⁷ Thorsson Kristinn R (1997) Layered modular action control for communicative humanoids *Proc Computer Animation 97* IEEE Computer Society pp 134-143
- ³⁸ Lee WS Kalra P Magnenat Thalmann N (1997) Model based face reconstruction for animation *Proc Multimedia Modeling (MMM) 97* Singapore (to appear)
- ³⁹ Moccozet L Magnenat Thalmann N (1997) Dirichlet free form deformations and their application to hand simulation *Proc Computer Animation 97* IEEE Computer Society pp 93-102
- ⁴⁰ Sederberg TW Parry SR (1986) Free-form deformation of solid geometry models *Computer Graphics (SIGGRAPH 86)* 20(4) pp 151-160 1986
- ⁴¹ Escher M Magnenat Thalmann N (1997) Automatic cloning and real-time animation of a human face *Proc Computer Animation 97* IEEE Computer Society, pp 58-66

⁴² Kalra P Mangili A Magnenat Thalmann N Thalmann D (1992) Simulation of facial muscle actions based on rational free form deformations *Proc Eurographics 92* pp 59 69

⁴³ Magnenat Thalmann N Kalra P Pandzic IS (1995) Direct face to face communication between real and virtual humans *International Journal of Information Technology* Vol 1 No 2 pp 145 157

⁴⁴ Philips Speech Processing URI address [http //www.speech.be.philips.com](http://www.speech.be.philips.com)

⁴⁵ Pure Speech URI address [http //www.speech.com](http://www.speech.com)

⁴⁶ Vocalis URI address [http //www.vocalis.com](http://www.vocalis.com)

⁴⁷ Capin TK Pandzic IS Noser H Magnenat Thalmann N Thalmann D (1997) Virtual human representation and communication in VLNET *IFFE Computer Graphics and Applications* March April 1997 pp 42 53

⁴⁸ Doenges Peter Lavagetto Fabio Ostermann Joern Pandzic Igor Sunday Petajan Eric MPFG-4 Audio/video and synthetic graphics/audio for real time *Interactive Media Delivery Image Communications Journal* Vol 5 No 4 May 1997

⁴⁹ Kalra P Becheiraz P Magnenat Thalmann N Thalmann D (1997) Communication between synthetic actors Chapter in the book *Automated Spoken Dialogues Systems* (Ed Luperfoy S) MIT Press Cambridge MA (to appear)

Articulated Figure Motion Capture, Dynamics and Control

Dimitris Metaxas
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104 6389
e mail dnm@central.cis.upenn.edu
http://www.cis.upenn.edu/~dnm

1 Introduction

The automatic generation of realistic articulated figure movement is a very important yet open problem in animation. The development of physics based modeling methods in the mid eighties has resulted in a variety of techniques towards this goal. However there is still a long way to go before this goal can be achieved. This part of the notes will present a variety of physics based modeling methods for capturing, animating and controlling human motion.

We first present an efficient dynamic constraint formulation for the animation of articulated figures. Even though the method is independent of the underlying dynamic formulation (e.g. Lagrangian or recursive) we have chosen a recursive formulation for efficiency. The method allows the formulation of any type of constraints and the control of articulated figures in terms of trajectory following.

Second we present a methodology for the motion capture of articulated figures from single/multiple cameras. The captured motion can then be coupled with the motion of other figures generated with recursive dynamics. Such techniques are very useful in virtual reality applications.

Finally we address the problem of dynamic simulations involving both open and closed loops and we apply it to the problem of a human climbing a ladder. Given the problem formulation we employ a Lagrangian as opposed to a recursive dynamics approach.

Together the above three techniques can be used for the successful creation of complex animations. For each of the above techniques we present the geometric, kinematic and dynamic formulation as well as a series of animations that demonstrate their power.

2 Recursive Dynamics and Efficient Dynamic Constraints for Animating Articulated Figures

Evangelos Kokkevis¹ and Dimitris Metaxas

¹ Alias|Wavefront, 614 Chapala Str, Santa Barbara, CA 93101. Email: vangelis@aw.sgi.com

Among the most interesting yet largely open research problems in computer animation is the generation of natural looking motion for articulated figures. Many researchers have been interested in developing techniques that are efficient enough to be used for interactive applications and yet have sufficient

power to handle complex figures and environments. Physics based simulation offers the potential to achieve realistic motion at the cost, though, of increased programming complexity and computational resources requirements. The development of fast dynamics simulation algorithms that allow motion to be easily created and controlled and that are not overly complicated to implement is a non-trivial problem.

A number of approaches for generating physically consistent motion of articulated figures can be found both in the mechanical engineering and recently in the computer graphics literature. The traditional engineering approach has always been to derive expressions for a figure's motion directly from Lagrange's equations through symbolic manipulation as described in [24]. For a figure with n degrees of freedom (*dofs*) the *reduced coordinates* Lagrangian formulation produces a coupled system of n equations with n unknowns. Although this formulation is powerful and can be used for any type of objects and constraints, it suffers from two major efficiency related drawbacks. First, for figures with a large number of *dofs*, the symbolic differentiation that produces the equations of motion becomes extremely complicated and time consuming. Second, and even more important, solving the resulting $n \times n$ system of equations takes $O(n^3)$ time, thus eliminating the possibility of achieving real-time rates for large values of n . Although this running time has been improved by a clever reformulation of the equations to $O(n^2)$ by Walker and Orin's [68] algorithm, but is still not efficient for the majority of high *dof* systems.

In the computer graphics literature, most forward dynamics algorithms presented use a *maximal coordinate* Lagrangian formulation. In contrast to the reduced coordinates counterpart where one parameter is used to represent each *dof* of the figure, maximal-coordinate formulations use six parameters for each articulated link. The maximal-coordinate formulation leads to more parameters than *dofs* and the superfluous parameters are eliminated using explicit constraint equations. Shabana [63] explains in detail how constraints are used to model the joints of an articulated figure. For computer graphics applications, explicit constraint methods have been used by Barzel and Barr [9] to attach rigid segments together, by Metaxas and Terzopoulos [43-44] to simulate rigid and non-rigid articulated figures, and by Witkin et al. [69] and Baraff [8] to simulate articulated figures with rigid links. Algorithms based on maximal-coordinate formulations typically have $O(n^3)$ running times for general articulated figures, although Baraff has shown that by using sparse matrix techniques, an $O(n)$ bound can be achieved. Sparse matrix techniques have also been used by other researchers to achieve lower computational complexity in dynamics simulations involving particular types of articulated figures [51].

Maximal coordinate methods with explicit constraints lead to modular and extensible systems where connections between objects can be made or broken on the fly with very small computational overhead. However, these methods are not effective in modeling simple joints commonly found in articulated figures. In general, an unconstrained rigid link has 6 *dofs* (3 rotational and 3 translational). Therefore, if the link is connected to the figure with a k *dof* joint, $(6 - k)$ constraint equations are needed in a maximal-coordinate formulation to eliminate the remaining *dofs*. A figure with n single *dof* joints will require $5n$ constraint equations to be solved at every simulation time step. In contrast, in a reduced coordinate formulation, only n parameters, one for each *dof*, would need to be solved for. Furthermore, expressing a specific joint type with a constraint formula is not particularly straightforward. Spherical joints can be modeled with a trivial point-to-point constraint, but more or less anything beyond that (e.g., prismatic joints, revolute joints with 1 or 2 *dofs*, cylindrical joints) requires

significantly more complex and hence computationally expensive constraint equations. In addition maximal coordinate based methods by using explicit constraints for modeling joints tend to suffer from drifting. The drifting caused by the inaccuracy of numerical integration manifests itself as separation of the figure's links during the course of the simulation and creates a physical inconsistency. Methods using reduced coordinates operate directly in the joint space of the articulated figure. In other words all the computations are naturally performed in terms of joint accelerations, velocities and angles. Maximal coordinate methods work in the Cartesian space and quantities need to be continuously converted back and forth between the Cartesian and the joint space. Frequent tasks such as computing a joint displacement, checking for joint limits, applying internal torques to joints, the conversion becomes a significant inconvenience.

Fortunately there exist efficient techniques based on the reduced-coordinate formulation that can be used for articulated figures. Featherstone [19], Armstrong and Green [3] and Bae and Haug [6] have developed recursive forward dynamics algorithms that run in $O(n)$ time. Armstrong's algorithm is limited to figures with spherical joints only. Featherstone's articulated body method is one of the fastest linear complexity algorithms for figures without closed kinematic loops. Moreover it trivially supports arbitrary joint types (translational, revolute, screw and their combinations) with one or more dofs. Featherstone in his work uses spatial vectors, a notation that incorporates translational and rotational components of physical quantities in a single six dimensional unit. Although the algebra of spatial vectors takes a bit of getting used to, the elegance of the compact equations soon becomes apparent. It is our opinion that the algorithm itself is not any harder to implement than other dynamics algorithms and the gain in performance and generality make the choice well worth it.

Featherstone's algorithm is based on the reduced coordinate formulation and does not rely on constraints to model joints. However the need for constraints goes much beyond modeling joints and the original algorithm by not providing any means to deal with them is seriously handicapped. To mention a few of the uses that constraints might have: modeling of closed kinematic loops, enforcing joint limits, simulating contact between a figure and its environment, controlling the figure's motion and connecting figures together. It is clear that constraints play a particularly important role in dynamics based simulation.

We will first present a method to extend Featherstone's method (see Appendix 2 for more details on this method) that allows incorporating one or more constraints on the simulated figures' motion. The algorithm we provide is not only simple to implement from the programmer's perspective, it is also computationally efficient and therefore maintains the interactive rates that the original method was capable of achieving. Furthermore, our constraint evaluation technique is completely automatic and does not place the burden on the user to manually tune any parameters to make it work.

Second we present a complete framework that integrates motion control and collision handling with fast forward dynamics simulation. In the literature there exists work dealing with the problems of motion control ([9, 59, 45, 28]) and collision response ([47, 7]) as separate topics. However it has not yet been shown how these different components can be made to work together. The system presented here achieves the integration of these components and becomes a powerful and efficient animation tool for generating and controlling interactive, realistic motion for articulated figures.

2.1 Introduction to Constraints

As we mentioned earlier Featherstone's recursive algorithm (see Appendix 2) does not explicitly use constraints to model the joints of an articulated figure. Instead the constraints that keep the links of the figure connected are implicit in the formulation. In these notes we deal with additional external constraints enforced on the figure. From now on when we talk about constraints we will be referring to these external constraints.

Constraints are used to enforce particular conditions on figures. More specifically the constraints we will be using in these notes are *acceleration constraints* conditions that concern the acceleration of either points on links of the figure or joints of the figure. We intend to show that the acceleration constraints are simple building blocks that can be used to achieve more complex higher level goals including modeling closed kinematic loops in articulated figures connecting figures together simulating impact and contact of dynamic objects enforcing joint limits and interactively controlling the motion of figures.

The acceleration constraints are enforced through *constraint forces* acting on the figure. One force corresponds to each constraint. The goal of a constraint evaluation algorithm is to determine what constraint forces will satisfy all the constraints simultaneously. The magnitude of each constraint force in general changes during the figure's motion and therefore needs to be evaluated at every time step of the dynamic simulation. In the following we will present and explain an evaluation algorithm that can compute constraint forces at every timestep of the dynamics simulation.

In its most general form the equation governing the motion of an articulated figure at any time instant is given by

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = F^B \quad (1)$$

where q is the vector of the generalized coordinates of the figure (in the reduced coordinate formulation this is the vector of all the joint angles of the figure) $M(q)$ is the mass matrix $C(q, \dot{q})$ are the generalized Coriolis forces and $G(q)$ are the gravity terms. Finally F^B is the vector of generalized external and actuator forces acting on the figure.

In a numerical simulation time is discretized and therefore forces applied to the figure have an instantaneous effect only on the figure's acceleration \ddot{q} . The velocity \dot{q} and position q change only after some time elapses. Subsequently to achieve a particular acceleration effect at some time instant t we can momentarily freeze time at t determine the force that when applied to the figure will result in the desired acceleration and then advance the time. We use this technique in the constraint evaluation algorithm we will present. In the following sections we will show how forces satisfying one or more constraints can be computed easily and efficiently. We will then show how acceleration constraints can be used to simulate the effect of impulsive forces on figures (such forces are commonly found in collisions) to enforce joint limits to compute contact forces between two colliding figures and to allow figures to follow user defined motion trajectories. We will finally show examples of how when all these features are integrated in one animation system realistic motion for articulated figures can be produced quickly and effortlessly.

2.2 Evaluating a Single Point Acceleration Constraint

In this section we present our constraint evaluation technique that works for a single one dimensional point acceleration constraint. We assume that we have at our disposal a forward dynamics algorithm (such as Featherstone's Articulated Body Method) that given an articulated figure and a set of forces acting on it it can compute the acceleration of the figure's links. Let us assume for now that we are given an articulated figure and a point P on one of the figure's links. Our goal is to determine a constraint force f^c which when applied to the figure will set the acceleration A_P of P equal to some desired value A_P^d . To make things less complicated we will deal with each of the three Cartesian components of the acceleration vector separately. Let a_i and a_i^d denote the the actual and desired accelerations respectively of point P along one particular direction i . From now on when we talk about the acceleration of P we will be referring only to the component of the acceleration along i . The constraint expression C that we use to set the actual acceleration to the desired value is

$$C = a_P - a_i^d \quad (2)$$

where a_i and a_i^d are the magnitudes of the accelerations a_i and a_i^d respectively. What we are looking for is a constraint force that when applied to the figure it will force equation

$$C = 0 \quad (3)$$

to be true. The most natural place to exert the constraint force is of course point P itself. In addition since we are interested in affecting the acceleration of the point along direction i the direction of the force will be along i as well. Therefore the problem has been reduced to that of determining the magnitude f^c of a constraint force f^c along i that when applied to P it will result in $a_P = a_i^d$.

Before we proceed to explain how the constraint force magnitude f^c is determined we need to mention two important facts (Note that these two facts hold only under the assumption that we have momentarily frozen the time at a particular time instant. Proofs of these facts are in Appendix 1)

- **Fact 1** The net effect¹ that multiple forces acting simultaneously on a figure have on the acceleration of a point is equal to the sum of the net effects that each force would have separately on the acceleration of that point, and
- **Fact 2** The net effect that a force has on the acceleration of a point on the figure is proportional to that force's magnitude

When no constraint forces are acting on the figure point P has some acceleration a_i^0 . This non-zero in general acceleration is due to all the external (including the gravitational) and internal forces acting on the figure. Let us for the time consider the case where a_i^d , the desired acceleration of P is zero. The correct constraint force f^c would have to counterbalance the effect of all the other forces in order to bring the point's acceleration to zero. If we could find a force f^c whose net effect is to give P an acceleration equal to $-a_i^0$ then Fact 1 confirms that f^c combined with the other forces will set

¹By the term *net effect* of a force we refer to the effect that particular force has on the figure when all the other forces (internal and external) are zero

the acceleration of P to zero. In general, for an arbitrary value of a_j^d , the constraint force should have a net effect on the acceleration of P equal to $a_p^d - a_p^0$. This constraint force, combined with the other forces acting on the figure, will set the acceleration of P to a_p^d .

The question is how does one go about determining what the net effect of a force is. If we were willing to do some mathematical manipulations, then we could symbolically derive the equation that relates the magnitude of a force applied at P to the force's net effect on the acceleration of P . In other words, we would have to perform an inverse dynamics calculation that would provide us with the desired constraint force. However, there is a much more practical way of achieving the same result that requires no symbolic manipulations and uses the already available forward dynamics algorithm. We first apply a force f^1 of unit magnitude ($f^1 = 1$) at P . Using the forward dynamics algorithm, we compute the magnitude a_p^1 of the resulting acceleration a_p^1 at P . The acceleration at P reflects both the effect of the force f^1 and the effect of the rest of the external and internal forces acting on the figure. We know that the acceleration at P due to the external and internal forces has magnitude a_p^0 , and therefore, using Fact 1, we can compute the net effect on the acceleration of P , a_p^1 , of f^1 from

$$a_p^1 = a_p^1 - a_p^0 \quad (4)$$

Using Fact 2, we can now easily compute the magnitude of the force f^c that will satisfy the constraint. If a force f^1 has a net effect on the acceleration of P equal to a_p^1 , then a force f^c that will achieve a net effect on the acceleration of P equal to $a_p^d - a_p^0$ will have magnitude

$$f^c = f^1 \frac{a_p^d - a_p^0}{a_p^1} \quad (5)$$

If we denote by C^0 the value of the constraint expression C when no constraint force is acting on the figure, then

$$C^0 = a_p^0 - a_j^d \quad (6)$$

and noticing that $f^1 = 1$, the constraint force from (5) can be written equivalently as

$$f^c = \frac{-C^0}{a_p^c} \quad (7)$$

To summarize, the constraint force f^c that will set the acceleration of P to some desired value a_j^d can be computed using the following three steps:

1. Set the external force at P to zero and use the forward dynamics algorithm (in our case, Featherstone's method) to compute the link accelerations. From the linear and angular acceleration of the link that contains P , compute the magnitude of the acceleration at P , a_p^0 , and evaluate the constraint expression C^0 .
2. Apply a unit force f^1 at P and use the forward dynamics algorithm to compute the magnitude of the resulting acceleration of P , a_p^1 . Compute the net effect a_p^1 of f^1 using (4).
3. Compute the required constraint force using (7).

2.2.1 Cost Analysis

Now let us derive the computational cost of the constraint force evaluation technique we just described. Although we will treat the forward dynamics algorithm as a black box that provides the figure's accelerations, our calculations are based on the assumption that we are using Featherstone's $O(n)$ Articulated Body method. One can easily see that from the three steps of the procedure we outlined above, only the first two have a significant computational expense. In the first step we compute the link accelerations without a constraint force, and in the second step we compute them given the unit force f^1 . Each of these calculations takes $O(n)$ time. Overall, the constraint evaluation algorithm takes $2O(n)$ time that is still $O(n)$. Therefore, one constraint does not increase the asymptotic complexity of the dynamic simulation.

2.3 Solving Multiple Constraints

In the previous section we presented an algorithm that evaluates a single scalar constraint. In this section we will show how this algorithm can be extended to handle multiple, simultaneously active constraints.

Assume that there are k scalar constraint equations of the same form as before that we need to satisfy simultaneously. In other words, for each i we need to find a constraint force f_i^c that will satisfy the corresponding constraint equation

$$C_i = a_{P_i} - a_i^d = 0 \quad (8)$$

Suppose that we wanted to use the algorithm from the previous section and try to solve each constraint sequentially. We would start by computing f_1^c , the constraint force that satisfies equation $C_1 = 0$. We would then proceed computing f_2^c for the second constraint, only to realize that when f_2^c is applied to the figure, the first constraint is no longer satisfied. The reason is that, in general, constraint force f_i^c affects not only constraint i but all the other constraints as well. Which means that we cannot solve each constraint separately and expect to combine together all the constraint forces we found. We need to take into account the effect that each constraint force has on all the other constraints and determine all the constraint forces simultaneously.

Computing the net effect that the constraint force associated with constraint j has on all k constraints C_i is performed the same way as before. We first compute $a_{P_i}^0$ and C_i^0 for $i = 1 \dots k$, the magnitude of the acceleration of points P_i and the value of the constraint expressions respectively when no constraint forces are acting on the figure. We then set the j th constraint force to be of unit magnitude $f_j^1 = 1$ and we compute the link accelerations using the forward dynamics algorithm. From the link accelerations we can compute the magnitude of the accelerations $a_{P_i}^{1,j}$ of all points P_i . For each i from 1 to k , we compute the net effect $a_i^{c,j}$ of the force f_j^1 on the acceleration of point P_i from

$$a_i^{c,j} = a_{P_i}^{1,j} - a_{P_i}^0 \quad (9)$$

By repeating this process for all k constraints, we can compute the k^2 coefficients $a_i^{c,j}$. Fact 1 tells us that when multiple constraint forces are acting together, their combined net effect is equal to the

sum of the net effects of each one. In addition, Fact 2 tells us that double the constraint force will have double the effect on the constraints and in general u units of the constraint force will produce u times the effect of one unit. Our goal is therefore to find how many units u_j of each constraint force we need to use to satisfy all the constraints simultaneously. We can express this as the following multidimensional problem:

$$\begin{aligned} u_1 a_1^{c_1} + u_2 a_1^{c_2} + u_3 a_1^{c_3} + \dots + u_k a_1^{c_k} &= -C_1^0 \\ u_1 a_2^{c_1} + u_2 a_2^{c_2} + u_3 a_2^{c_3} + \dots + u_k a_2^{c_k} &= -C_2^0 \\ &\vdots \\ u_1 a_k^{c_1} + u_2 a_k^{c_2} + u_3 a_k^{c_3} + \dots + u_k a_k^{c_k} &= -C_k^0 \end{aligned} \quad (10)$$

If we define

$$\mathbf{A}' = \begin{pmatrix} a_1^{c_1} & a_1^{c_2} & \dots & a_1^{c_k} \\ a_2^{c_1} & a_2^{c_2} & \dots & a_2^{c_k} \\ \vdots & \vdots & \ddots & \vdots \\ a_k^{c_1} & a_k^{c_2} & \dots & a_k^{c_k} \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} -C_1^0 \\ -C_2^0 \\ \vdots \\ -C_k^0 \end{pmatrix} \quad \text{and} \quad \mathbf{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_k \end{pmatrix}$$

then we can replace (10) with

$$\mathbf{A}' \mathbf{u} = \mathbf{v} \quad (11)$$

By solving the above system for u_1, \dots, u_k we compute the constraint forces whose magnitude is

$$f_j^c = u_j f_j^1 \quad (12)$$

for all $j = 1, \dots, k$.

2.3.1 Singular Systems

Since the constraints we attempt to enforce on the figures are arbitrary, there is no guarantee that the coefficient matrix \mathbf{A}^c will be non-singular. In fact, if constraints are linearly dependent (i.e., we have overconstrained the figures) the system will become singular and will have either an infinite number of solutions or no solution at all. However, this is not as big of a problem as it sounds. What we are really interested in is finding *one* valid set of constraint forces that will satisfy all the constraints when possible. Among the various techniques that exist for solving singular systems, we have chosen to use Singular Value Decomposition (SVD) to deal with the case of overconstrained systems. The big advantage of SVD is that it computes the solution vector \mathbf{u} that has the smallest possible norm. What this means in our case is that if there are multiple solutions, the solution that uses the smallest constraint forces will be chosen. Small constraint forces will always give a more numerically stable solution than large forces. Alternatively, if the system has no solution, then a set of forces that satisfies all the constraints to the best degree possible will be chosen. Such a situation would occur when two constraints are fighting against each other and it is impossible to satisfy them both at the same time.

2.3.2 Dynamics Pseudocode

In this section we present the pseudocode of a program that performs the dynamic simulation in the presence of external constraints. The reader should notice how short the code is and how once a forward dynamics simulator is available very little effort is required to add the constraint evaluation scheme. The main execution loop of the simulation process is as follows:

```

1   procedure DynamicSimulation(startI endI  $Q_{start}$   $Q_{start}$ )
2   (* set the initial state of the system *)
3    $Q \leftarrow Q_{start}$ 
4    $\dot{Q} \leftarrow \dot{Q}_{start}$ 
5   time  $\leftarrow startI$ 
6   (* simulation loop *)
7   while(time < endI)
8        $F^I \leftarrow \text{ComputeOtherForces}(Q, \dot{Q})$ 
9        $F^C \leftarrow \text{ComputeConstraintForces}(F^I, Q, \dot{Q}, C)$ 
10       $Q \leftarrow \text{ComputeLinkAccelerations}(F^I, F^C)$ 
11      Integrate( $\Delta t$ ,  $Q$ )
12      time  $\leftarrow time + \Delta t$ 

```

In the above code fragment Q is the vector of generalized coordinates that are used to describe the articulated figures in the environment. The elements of Q are typically the joint angles of each figure and the position of each figure's root link in space. The vector Q together with its first derivative \dot{Q} describe the state of the dynamically simulated figures. Vectors Q_{start} and \dot{Q}_{start} define the initial state of the figures at time *startI* that the simulation begins. Procedure **ComputeOtherForces** calculates all the external (e.g. gravitational) and internal (e.g. actuator torques) forces F^I active on the figures. Procedure **ComputeConstraintForces** computes the constraint forces acting on the figure using the method we described in the previous section. Assuming that there are k constraints of the form shown in (8) then the pseudocode for the procedure can be written as:

```

1   procedure ComputeConstraintForces( $F^I, Q, \dot{Q}, C$ )
2   for  $i = 1$  to  $k$ 
3        $F^C[i] \leftarrow 0$ 
4    $Q^0 \leftarrow \text{ComputeLinkAccelerations}(F^I, F^C)$ 
5    $A^0 \leftarrow \text{PointAccelerations}(Q^0)$ 
6    $C^0 \leftarrow \text{EvaluateConstraints}(A^0)$ 
7   for  $i = 1$  to  $k$ 
8        $F^C[i] \leftarrow 1$ 
9       if ( $i <> 1$ ) then  $F^C[i-1] \leftarrow 0$ 
10       $Q^1 \leftarrow \text{ComputeLinkAccelerations}(0, F^C)$ 
11       $A^1 \leftarrow \text{PointAccelerations}(Q^1)$ 
12      for  $j = 1$  to  $k$ 
13           $A^C[j][i] \leftarrow A^1[j] - A^0[j]$ 

```

```

14      $v[i] = -C^0[i]$ 
15      $F^C \leftarrow \text{SolveLinearSystem}(A', v)$ 
16     return  $F^C$ 

```

Lines 2 through 6 evaluate the constraint expressions when all the external forces we apply are set to zero. Procedure **ComputeLinkAccelerations** is in essence the forward dynamics algorithm that given all the forces acting on the figures computes the accelerations of the figures links Q . In our particular implementation we are using Featherstone's algorithm but in general any forward dynamics algorithm could be substituted. The acceleration of all the points P_i is calculated from **PointAccelerations**. The point accelerations are passed to procedure **EvaluateConstraints** that simply computes the value of the constraint expressions from (8). In lines 7 through 14 a unit test force is applied for each constraint and the resulting link accelerations are computed. This way the constraint system matrix A' is filled column by column. Once the system is formed line 15 calls a linear system solver to compute the vector of the constraint forces F^C .

Once the constraint forces have been evaluated the **DynamicSimulation** procedure in line 10 computes the link accelerations when all the forces including the constraint forces are active. These accelerations are guaranteed to satisfy all the k acceleration constraints when the constraint system has a solution. It remains to perform an integration compute the new state of all the simulated figures and advance to the next time instant. For the sake of simplicity in the pseudocode we have assumed that a constant integration timestep Δt is used. However we highly recommend the use of adaptive timestep integration both for numerical stability and speed.

We urge the reader to notice at this point how simple it is to add this constraint force evaluation procedure to a forward dynamics simulation system. The **ComputeLinkAccelerations** procedure is already available from the simulator and therefore the only significant new piece of code that needs to be supplied is code for solving the linear system. Such code can readily be found in a number of sources. In our implementation we have used an LU decomposition algorithm to solve non singular systems and an SVD algorithm for singular systems as described in [57].

2.3.3 Cost Analysis for Multiple Constraint Evaluation

It was shown earlier that with one external constraint the dynamic simulation still maintains an $O(n)$ running time. Unfortunately neither our algorithm nor any other algorithm we are aware of can achieve linear time performance when multiple arbitrary external constraints are simultaneously active. However the situation is not exactly hopeless. We will show in this section that the performance penalty for having multiple constraints is not unreasonably large and interactive simulation rates can still be attained.

Let us analyze the cost of the procedure **ComputeConstraintForces** assuming that the simulated figure has n links and that there are k active constraints. Since the forward dynamics algorithm we use has linear time performance each call to **ComputeLinkAccelerations** takes $O(n)$ time. The procedure **EvaluateConstraints** evaluates k simple expressions and clearly takes $O(k)$ time. Therefore lines 2 through 6 take $O(n) + O(k)$ time to execute. Within the loop (lines 7 through 14) line 10 is executed k times with a cost $O(n)$ each time and line 13 is executed k^2 times with a cost $O(1)$ each time. Hence the overall cost of the loop is $O(kn) + O(k^2)$. Finally the running time for solving the linear system

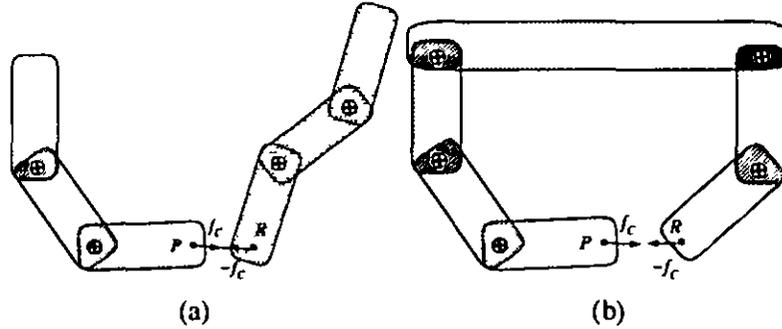


Figure 1 Attaching one point to another (a) Attaching two figures and (b) Creating a closed loop within the same figure

in line 14 is $O(k^3)$ using either LU decomposition and SVD. Adding up these running times we get the overall cost of the procedure which is $O(kn) + O(k^3)$. Although the $O(k^3)$ term seems to be prevailing in most cases the number of constraints k is much smaller than the number of links of the simulated figures n . In these cases the $O(kn)$ term dominates and therefore the algorithm has a very satisfactory performance. To our knowledge a system of k constraints on n dofs cannot be solved in less than $O(kn) + O(k^3)$ time. Our technique therefore compares favorably to the other fast techniques while still being extremely simple to implement.

2.4 Beyond Point Acceleration Constraints

So far in our discussion we have used a single constraint type: a constraint that sets the acceleration of a point on the figure to a particular value as expressed by (2). However, our algorithm is not limited to only this type of constraint. In the following sections we will show how constraint expressions that involve accelerations of multiple points can be used. We will also show that the same way point accelerations are used to constrain the motion of a point on the figure, joint angle accelerations can be used to constrain the motion of a joint. Finally, we will demonstrate how the constraint evaluation procedure can be easily modified to compute impulsive constraint forces that alter velocities instead of accelerations.

2.4.1 Attaching One Point to Another

Imagine we wanted to constraint two points P and R to move together. These points might belong to links of the same figure (Fig. 1(b)) or to links of two different figures (Fig. 1(a)). The expression that measures the difference between the accelerations of the two points is

$$C = a_P - a_R \quad (13)$$

and by enforcing the condition $C = 0$ we can ensure that points P and R will not separate (assuming that P and R are originally coincident and have the same velocity). Such a constraint can be handled by the algorithm we described above with one small modification. The constraint force is not an external

force anymore but it is an internal force acting from the one point to the other. Therefore according to the law of action and reaction the constraint force should be applied in pairs. When testing with the unit force f^1 if f^1 is applied to P $-f^1$ should be applied to R . After the system is solved the constraint force f^c should be applied to P and $-f^c$ should be applied to R . Apart from this change the constraint evaluation algorithm remains the same.

If the points P and R belong to links of the same articulated figure then by constraining them to move together we create a *kinematic loop*. Simulating efficiently the dynamic motion of figures with closed kinematic loops is a hard problem that the original algorithm of Featherstone cannot address. Featherstone in his book [19] describes a technique to deal with closed kinematic loops which he admits that is only of theoretical interest since it is too inefficient to be practically useful. A good overview of alternative techniques can be found in the work of Lilly [38].

2.4.2 Joint Constraints

Sometimes it is important to have direct access to the motion of a joint instead of the motion of a point of the figure. Joint acceleration constraints can be specified in a fashion similar to point acceleration constraints. For a particular joint variable q if the expression

$$C = q - q^d \quad (14)$$

is used then enforcing the constraint $C = 0$ will set the joint acceleration q equal to a desired value q^d . The constraint force (or torque) that satisfies a joint constraint is actually internal and acts directly on the joint through the joint's actuator. For joint constraints instead of measuring point accelerations we measure joint accelerations. Apart from this consideration no changes need to be made to the constraint evaluation code to accommodate joint constraints. In our implementation since reduced coordinates are used the joint accelerations \dot{Q} are computed directly from the forward dynamics algorithm and therefore evaluating expression (14) is just a matter of subtracting two values.

2.4.3 Impulsive Constraints

So far we have dealt with the problem of computing constraint forces that will set the accelerations of points or joints of a figure to a particular desired value. These constraint forces need to be computed at every time step of the dynamic simulation and be applied to the figure before the new link accelerations are computed from the forward dynamics algorithm. In this section we will deal with a slightly different problem that of computing the effect of *impulsive forces* on a figure. There are times when during the course of a simulation a large force of very short duration is exerted on a figure and causes an immediate change in the figure's state. The effect of an impulsive force can be considered as an instantaneous change in the figure's velocity. A typical example of such a force is the impulsive force due to the impact of a figure with a stationary object. When a figure's link collides with the stationary object the link's velocity changes in an infinitesimally short time as a result of the collision impulse.

If we were dealing with non-articulated figures then the effect that an impulse P has on the figure's velocity V would be easily computed through Newton's equation $P = M\Delta V$. However for articulated figures things are a bit more complicated since the impulse affects the velocities \dot{Q} of all

the dofs of the figure Moore and Wilhelms [47] have presented a method that iteratively constructs a linear system that can be used to compute the effect of impulses on articulated figures. The size of their system for a figure with n links is $6n \times 6n$ and requires $O(n^3)$ time to be solved. For complex figures such an algorithm would slow down significantly the simulation.

In this section we will show how by using the algorithm we developed for the acceleration constraints we can compute the effect of one or more impulsive forces as efficiently as we compute an equal number of acceleration constraint forces.

Assume we know that at a particular time instant the velocity of a point P on the figure changes instantaneously from its current value v_P^- to some known value v_P^+ due to an impulsive force acting on P . We want to find the effect of the impulsive force on the velocities of all the figure links. In other words if we know the velocities Q^- right before the impulse we want to find the velocities Q^+ right after the impulse. Let the infinitesimally small time when the velocity change takes place be δt . During this time interval the magnitude of the velocity of P changes from v_P^- to v_P^+ , and therefore the magnitude of the acceleration of P during the impulse is

$$a_P^{imp} = \frac{v_P^+ - v_P^-}{\delta t} \quad (15)$$

Similarly each joint variable q will have an acceleration q^{imp} during the impulse that will change its velocity from q^- to q^+ . Without loss of generality we can assume that the velocity change takes place over an interval of one unit of time ($\delta t = 1$). The acceleration of P during that time is equal to the change in the point's velocity $a_P^{imp} = (v_P^+ - v_P^-)$. We use a_P^{imp} as the magnitude a_P^d of the desired acceleration of P in the constraint expression (2) and employ the evaluation algorithm of section 2.2 to calculate the constraint force f^c . With f^c acting on the figure the acceleration of P will be equal to a_P^{imp} and after one unit of time elapses the velocity of P will change from v_P^- to v_P^+ . By applying f^c to the figure we compute the resulting acceleration q^{imp} of each joint variable q . Using the joint velocities q^- before the impulse and the accelerations q^{imp} we compute the joint velocities q^+ after the same unit of time elapses. The velocities q^+ are the post impulse velocities that we originally set to compute.

The procedure to compute the effects of an impulse to the joint velocities of a figure we just presented can be generalized to handle multiple simultaneous impulses. The algorithm that calculates the effects of the impulsive forces that change the velocities of k points from $v_I^- = [v_{I_1}^-, v_{I_2}^-, \dots, v_{I_k}^-]$ to $v_I^+ = [v_{I_1}^+, v_{I_2}^+, \dots, v_{I_k}^+]$ is

```

1   procedure ComputeImpulseEffect(Q-, vI+, vI-, FI, Q, Q)
2   for i = 1 to k
3       C[i] ← aIi - (vIi+ - vIi-)
4       Fc ← ComputeConstraintForces(FI, Q, Q, C)
5       Qd ← ComputeLinkAccelerations(FI, Fc)
6       Q+ = Q- + Qd
7   return Q+

```

where F^I is the vector of all the other forces acting on the figure at the time the impulses take place and Q^- and Q describe the current state of the figure. It is clear from the above code that the computational

cost of `ComputeImpulseEffect` is of the same order of magnitude as the cost of `ComputeConstraint Forces`

2.4.4 Inequality Constraints

There are times when the simple acceleration constraints we described so far do not adequately describe effects we wish to achieve. For example, modeling the interaction between a figure that is in contact with another object in the environment typically involves a set of two inequality and one equality constraint expressions. There is one inequality constraint expression for the relative acceleration at the contact point of the form $a_p \geq 0$ and one for the constraint force of the form $f^c \geq 0$. The first constraint expression ensures that the figure and the object in contact do not continue moving towards each other. The second constraint expression ensures that the contact force is repulsive. A third constraint expression of the form $a_p \cdot f^c = 0$ guarantees that either the acceleration of the point or the contact force are zero, a condition necessary for contacts.

Certain modifications to the constraint force evaluation algorithm that we presented are necessary to allow the handling of inequality constraints. Researchers have shown that such constraints can be handled by transforming the constraint problem into a Linear Complementarity Problem (LCP). Baraff in [7] has used an LCP based approach to compute contact forces between non-articulated objects. An excellent and complete coverage of how to determine contact forces using the LCP formulation can be found in [55]. Solving an LCP problem amounts to solving the system of (11) when one or more of the constraints involved are inequality constraints. Since Linear Complementarity based techniques have been discussed in the past we will not go into details on how to implement them. However, we should note that an LCP solver can be incorporated seamlessly to the system we already described. The basic method to compute the constraint system matrix A' we presented in section 2.3 is still used to setup the constraint system. After A' is determined, the specialized LCP solver is employed to compute the constraint forces.

2.5 Using Constraints

In this section we will describe how, using the basic constraint types we presented earlier, one can simulate collisions between figures and enforce kinematic joint limits. Furthermore, we will describe how the acceleration constraints can be used to provide a user with direct control over the simulated motion of the articulated figures.

2.5.1 Simulating Collisions

Simulating collisions typically involves two stages: the *detection* stage and the *response* stage. During the detection stage, the position and orientation of the figures' links is used to determine whether the polygonal surfaces of two links intersect. If this is the case, then some action needs to be taken to prevent the figures from further penetration. In a dynamically simulated environment, a collision response algorithm computes the necessary repulsive forces that prevent the surfaces from penetrating each other.

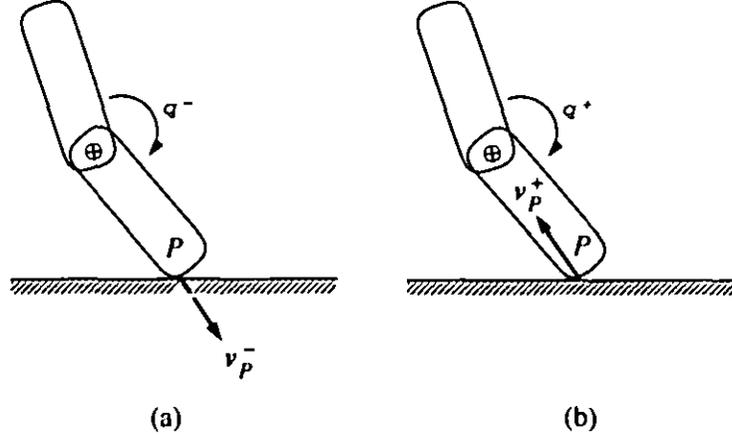


Figure 2 Collision between an articulated figure and a stationary object Snapshot (a) represents the figure right before the impact and (b) right after the impact The impact has an instantaneous effect only on the velocity but not on the position of the figure

To detect collisions in our system we use *I COLLIDE* a very efficient collision detection algorithm for convex polygonal objects devised by Cohen et al [31] One of the big advantages that *I COLLIDE* offers is that it detects which features (nodes edges and faces) of the two objects are the closest before the object geometries actually interpenetrate In order to use this information to setup our constraint problem we define a threshold distance between objects below which we consider the objects to be colliding In other words we enter the collision response stage when the two objects are sufficiently close but their geometries have not yet penetrated each other Since collisions of non convex polygonal objects cannot be directly handled by *I COLLIDE* we decompose non convex geometries to a set of convex geometries before we use them in the collision detection algorithm

We further divide the collision response into two phases the *impact* phase and the *contact* phase When a new collision point is detected between two objects an impulsive constraint is set up for that point and the impact phase is entered (Fig 2) The impulsive constraint sets the relative post-impact velocity v_i^+ between the two objects at the point of collision to

$$v_i^+ = -\epsilon v_i^- \quad (16)$$

where v_i^- is the relative velocity between the two objects at the collision point before the collision and ϵ is the coefficient of restitution a number in the range $[0 \quad 1]$ Since there might be other points that are already in contact from previous collisions we set up a similar impulsive constraint for each one of them Typically for these points v_i^- will be zero and therefore v_i^+ will be zero too However it is important to take into consideration these constraints too because otherwise the impact at the new collision point might result to penetration at the old contact points We can now use the procedure **ComputeImpulseEffect** to calculate the post impact velocities for all the links of the two colliding figures Once the velocities have changed the contact phase of the response algorithm is entered In this phase we set up inequality contact constraints for each pair of points of the colliding figures whose

distance is below the collision threshold. These constraints make sure that the two figures do not move any closer together by setting the desired relative acceleration between the contact points to be greater or equal to zero. As described in section 2.4.4, the constraint force f^c that satisfies each of these constraints will have to be repulsive or in other words non-negative. For contact constraints, an LCP based solution is required to guarantee that both the force and the acceleration conditions are satisfied.

In the past, researchers in computer graphics have simulated collisions mostly by using stiff springs and dampers at the contact points of the colliding objects [59, 45, 28]. Spring damper techniques have the advantage of being simple to implement but they often introduce instabilities to the numerical integration. Furthermore, to obtain good results, the elasticity and damping parameters need to be manually tuned by the user through a tedious trial and error process. Our collision response technique does not suffer from these numerical instabilities and more importantly does not require tuning by the user to work properly.

2.5.2 Simulating Joint Limits

A joint limit defines an angular displacement beyond which a joint is not allowed to move. In our system, joint limits are handled similar to collisions. In fact, enforcing joint limits is more efficient than simulating collisions. The reduced coordinate dynamic formulation used by Featherstone's algorithm provides direct access to the joint variables, making detection and response a straightforward task.

When a joint angle first reaches its limit, an impact phase is entered and an impulsive constraint for the joint angle is set up. Typically, we require the post-impact velocity of the joint to be zero. The same way the original point acceleration constraint algorithm was modified to handle joint acceleration constraints in section 2.4.2, the impulsive constraint algorithm for points can be modified to deal with joints. The impulse that instantaneously changes the velocity of the joint is now a torque applied at the joint itself in a direction opposing the motion against the limit. Once the impulsive constraint is evaluated (using a variant of `ComputeImpulseEffect` that handles joint velocities instead of point velocities), an inequality acceleration constraint can be set for the joint. The acceleration of the joint will either be zero or will have a direction pointing away from the joint limit. Similar to the contact constraints, the constraint torque that will satisfy the joint limit constraint should only be pushing away from the joint limit and not pulling towards it. Once again, an LCP based solution is required to compute the correct joint torques that will enforce the joint limits.

2.5.3 Following Motion Trajectories

In this section, we will show how the acceleration constraints can be used to make a particular point P of a figure follow a certain motion trajectory. Suppose we wanted P to move from its current position p^c to some desired position p^d . We will again deal with each one of the three components of the motion of P separately. Assume that along direction i , the current displacement of P is p_i^c , where $p_i^c = p^c \cdot i$ and its velocity is v^c . If the desired displacement of P along i is p_i^d , then we can use the following expression for the desired acceleration of P :

$$a_p^d = (p^d - p^c)K_p + (v^d - v^c)K_d, \quad (17)$$

where k_p and k_d are two constant coefficients otherwise known as *gains* and v^d is the desired velocity along the trajectory. The desired acceleration a_j^d can be used as a constraint that is enforced using the constraint force evaluation procedure we presented. At every step of the simulation procedure the new values obtained for p^c and v^c are used in (17) to obtain a new desired acceleration a_j^d . Equation (17) describes the acceleration of a second order system similar to a mass connected to a spring and damper. Although this expression reminds of the familiar PD control scheme used extensively in the robotics and graphics literature it bears one a major difference. A PD controller is used to compute the magnitude of a force, not an acceleration. The force computed by the PD controller will only attempt to give the system a particular acceleration, but unless the inertial properties of the system are accurately known, the actual acceleration of the system will depend heavily on the choice of k_p and k_d . The advantage of computing the desired acceleration instead of a force is that accelerations give precise control over the motion. The constraint evaluation scheme that we described guarantees that whenever possible the acceleration we request will be achieved.

Since we are dealing with physical systems, objects do not move instantaneously. Therefore it will take some time for P to get from p^c to p^d . How much time depends on the choice of the two gains k_p and k_d . The path that P follows to get from its initial position to its goal position also depends on the gains. If an *underdamped* motion is chosen, then P will quickly move to its desired position and then continue moving past it (overshoot) and oscillate around it for a while. For the purpose of trajectory following, the best strategy is to use a *critically damped* motion, where P moves as quickly as possible to the desired position with minimal overshoot. This type of motion can be achieved by setting the gains to

$$k_p = \frac{1}{t_s^2} \text{ and } k_d = \frac{\sqrt{2}}{t_s} \quad (18)$$

where t_s is called the *settling time* and reflects the amount of time it will take for the point to get to within 10% of its desired position. The settling time is a parameter that can be adjusted by the user depending on the particular effect that is desired. Smaller settling times will result in quicker response to a change in the desired position and higher settling times will result in more relaxed looking motion.

Instead of using a single point as a desired position, a complete predefined trajectory can be used. A trajectory consists of a number of points over time. Depending on the current time of the simulation, the respective desired point can be used for p^d in the expression of (17). This way, point P will be driven to follow a motion path that resembles (to a degree depending on the settling time) the predefined trajectory.

The same technique can be used to make a joint follow user defined joint angle trajectories by setting up joint acceleration constraints instead of point acceleration constraints. Furthermore, multiple trajectories can be followed by different points (or joints) of one figure at the same time. Using the trajectory following technique we described, a user can have easy control over the motion of any dofs of the figure he or she chooses to.

2.6 Stabilizing Constraints

So far we have dealt with acceleration constraints of the form $C(\mathbf{q}) = 0$. We presented an algorithm that computes the necessary constraint forces which guarantee that the specified constraint equations on the figure's accelerations will be satisfied. However, a number of common types of constraints do not appear in the form of an equation involving only acceleration terms. For instance, a simple *nail* constraint that forces a particular point P on a figure to stay at a desired location in space is described naturally by an equation of the form

$$C = \mathbf{p}^c - \mathbf{p}^d = 0 \quad (19)$$

where \mathbf{p}^c is the actual and \mathbf{p}^d the desired position of P . If P is initially at its desired position then, in theory, an acceleration constraint of the familiar form of equation (2) $C = \mathbf{a}_P = 0$ will keep P from moving. In practice, however, it is well known that the integration of such constraints leads to numerical instabilities. Sooner or later in the simulation P will drift away from its desired position.

To remedy this problem, we apply a method proposed by Baumgarte that stabilizes the constraints through linear feedback control [10, 72]. The method replaces equation (2) by a damped second order differential equation that has the same solution but is asymptotically stable

$$\mathbf{C} + 2\alpha\dot{\mathbf{C}} + \beta^2\mathbf{C} = 0 \quad (20)$$

In the example of the nail constraint, if C is given from (19) then $\dot{C} = \mathbf{v}_P$ and $C = \mathbf{a}_P$. The resulting stabilized constraint equation looks like

$$\mathbf{C} = \mathbf{a}_P + 2\alpha\mathbf{v}_P + \beta^2(\mathbf{p}^c - \mathbf{p}^d) = 0 \quad (21)$$

From the above equation we can obtain an expression for the desired acceleration at P , \mathbf{a}_P^d

$$\mathbf{a}_P^d = -2\alpha\mathbf{v}_P - \beta^2(\mathbf{p}^c - \mathbf{p}^d) \quad (22)$$

Notice how the above equation has the same form as (17) we used to control the figure's motion. By choosing $\alpha = \beta$ in the above expression we obtain the critically damped solution that can be used for fast stabilization.

The Baumgarte stabilization method can be used for any constraint expression C . We take the first and second derivatives of C and separate the acceleration, the velocity, and the position terms $\ddot{\mathbf{T}}$, $\dot{\mathbf{T}}$, and \mathbf{T} respectively, to form an equation of the form

$$\ddot{\mathbf{T}} + 2\alpha\dot{\mathbf{T}} + \beta^2\mathbf{T} = 0 \quad (23)$$

It is important to remember that during the simulation we use the values obtained for the velocity and position terms in the above expression from the previous integration timestep to obtain the value for the desired acceleration for the current timestep.

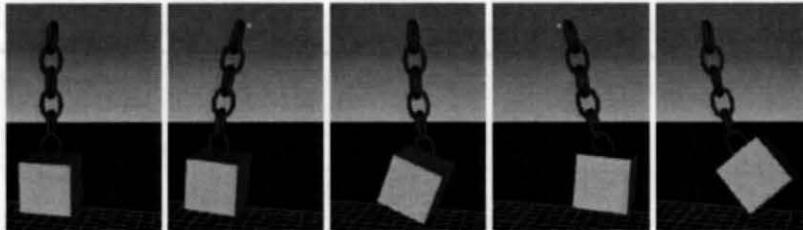


Figure 3: Snapshots from an interactive animation. The top of the chain is constrained to follow the motion of the square marker that is controlled by the user via the mouse.

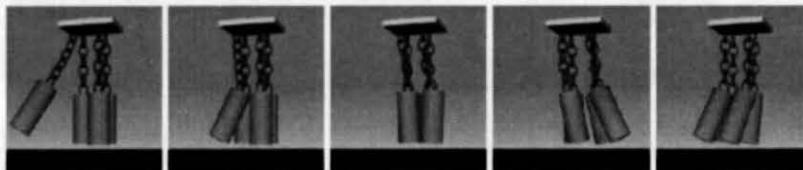


Figure 4: Snapshots from an animation involving collisions. Collisions are detected automatically and are handled by applying acceleration constraints at the collision points.

2.7 Experimental Results

We have used the algorithms we described above to generate a number of animations involving articulated figures and we were able to achieve interactive, near real-time rates (on a 200MHz MIPS 4000 workstation) for figures of moderate complexity, even in the presence of multiple constraints.

Figure 3 shows snapshots of an interactive animation of a chain with 6 links and a total of 16 dofs. Chain links are connected through 2 dof joints and the load at the bottom of the chain is linked with a spherical joint. Three one-dimensional constraints are used to control the motion of the top of the chain as the user interactively (using the mouse) sets its desired path by moving the cube. The settling time for this animation was set to $0.1sec$ and one can notice how the top of the chain lags behind the cube. If the cube stopped moving, the chain would catch up in approximately $0.1sec$ of simulation time.

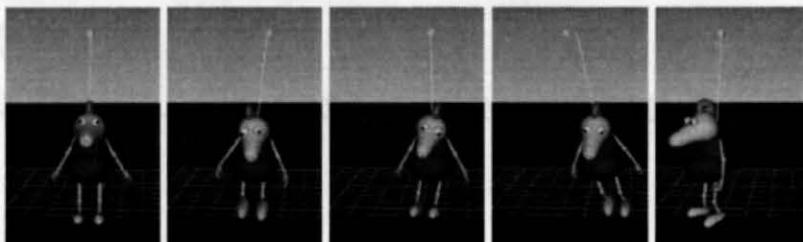


Figure 5: An interactive animation of an articulated character. The user controls the motion of the string attached to the character. Joints in the figure range from 1 dof to 3 dofs and joint limits are observed using constraints. Collisions are detected between the arms, the body and the legs of the figure.

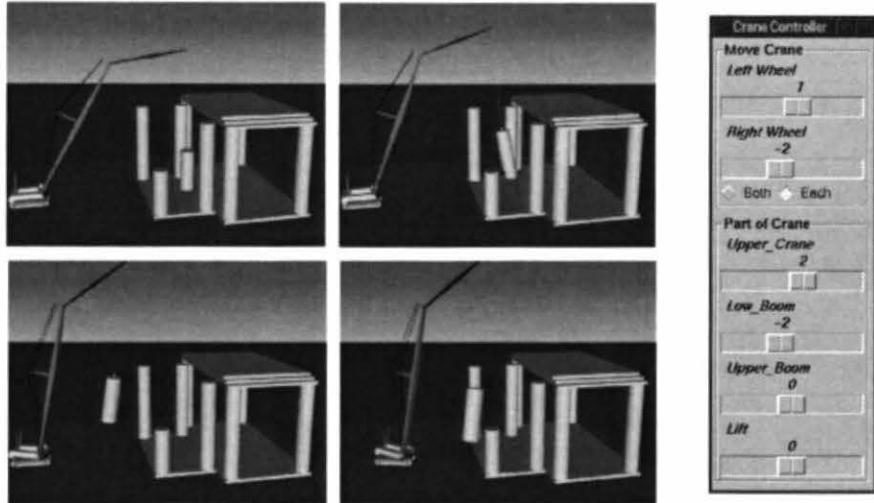


Figure 6: Snapshots from an interactive crane simulation. The crane is modeled as an articulated figure and is controlled by the user through the control panel shown. The dynamic simulation generates realistic motion in real time for the crane and the heavy load (10 tons) attached to the end of the crane's rope. Collisions are detected between the load of the crane, the booms and the environment

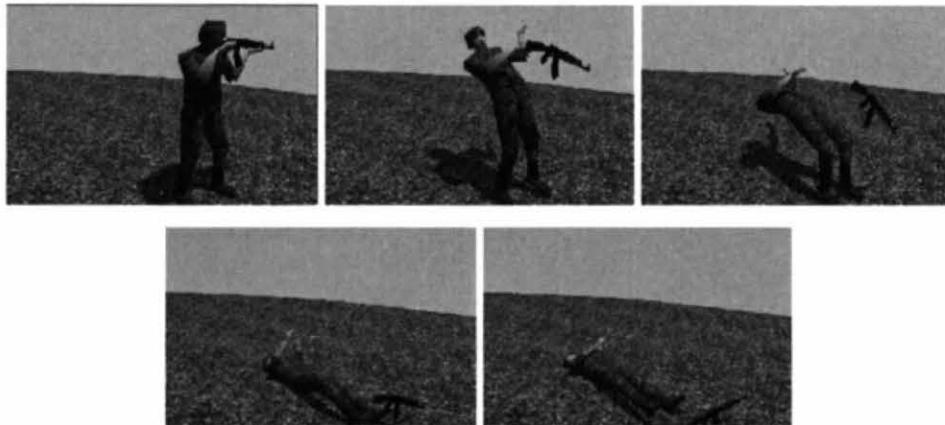


Figure 7: A wounded soldier: The soldier receives a large impulse at his upper chest and falls on the ground. Collisions between the soldier and the ground are automatically detected and handled by our system.

Figure 4 shows frames from an animation involving four chains with a total of 64 dofs. Collisions between the chains are automatically detected. The impact and contact stages of the collisions are modeled using the constraint algorithm described earlier. This simulation runs about 3 times slower than real time.

Figure 5 shows snapshots of an interactive animation of a more complex articulated figure. The cartoon character Ruffy has 21 dofs. Its joints have from 1 (eg. knees) up to 3 (eg. shoulder) dofs. Joint limits are set for all the joints (for example the knees do not bend forward past the vertical, the head does not rotate 360 degrees around the body, etc.) and are enforced using constraints. As in the example with the chain, the top of the string that drives Ruffy is constrained to the motion of the cube marker that is controlled by the user via the mouse.

The snapshots of figure 6 show four steps of an interactive crawler crane simulation and the panel through which the user controls the motion of the crane (left and right crawler velocity, cabin rotation, lower and upper boom angles). The motion of the heavy (10 tons) weight attached to the end of the crane's cable is generated in real time, thus giving the user realistic visual feedback of his actions. The crane is modeled as an articulated figure with 3 joints (cabin, lower boom, upper boom). Furthermore, the cable is connected to the end of the upper boom with a spherical joint and the load is connected to the other end of the cable with another spherical joint. A translational joint is placed at the top of the cable segment to allow the user to lift/lower the load. Realistic mass values are assigned to each segment of the crane and collisions are detected between the load, the booms and the environment. We are planning to incorporate this interactive crane simulation into an immersive Virtual Reality environment. The system will be used as a training tool that will allow crane operators to practice before moving to the much more expensive and dangerous field training.

The snapshots of figure 7 show a dynamic simulation of a complete human body modeled as an articulated figure. The body receives a large impulse on the chest and is left to fall on the ground. Collisions between the body parts and the ground are automatically detected and dealt with by our system. The running time of this simulation is about 1 frame per second.

It is important to note that in all the animations we have generated with our system, the user input has been limited only to specifying the motion trajectories. The user did not need to know anything about the internal implementation of the algorithms and was not required to adjust any parameters to get the simulations to work correctly.

2.8 Appendix 1 Proofs of Facts 1 & 2

Equation (1) is the general form of the equation of motion for an articulated figure. Let us assume that we freeze time at some particular instant t . As we mentioned earlier, any force applied to the figure will have an instantaneous effect only on the acceleration and not on the velocity or position of the figure. If a force f_P^f is applied at some point P on the figure, then (1) becomes

$$M(q)q'' + C(q, q') + G(q) = F + J_P^f(q)f_P^f \quad (24)$$

where F is the vector of all the known generalized forces (such as the link weights, internal joint torques, etc.) on the figure and $J_P^f(q)$ is the Jacobian matrix for point P . Setting the external force f_P^f to zero

we compute the acceleration \mathbf{q}^0 of the system due to all the other forces \mathbf{F}

$$\mathbf{M}(\mathbf{q})\mathbf{q}^0 + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \mathbf{F} \quad (25)$$

By subtracting (25) from (24) we get

$$(\mathbf{q}^x - \mathbf{q}^0) = \mathbf{M}^{-1} \mathbf{J}_I^I \mathbf{f}_P^x \quad (26)$$

where $(\mathbf{q}^x - \mathbf{q}^0)$ is the net effect that the force \mathbf{f}_P^x has on the accelerations of the figure. Equation (26) shows that the net effect a force has on the accelerations of the figure is proportional to the magnitude of the force. This observation extends to the acceleration of point P too by noticing that the velocity \mathbf{V}_P of point P is given by

$$\mathbf{V}_P = \mathbf{J}_P(\mathbf{q})\mathbf{q} \quad (27)$$

By differentiating the above equation we obtain the acceleration of P \mathbf{A}^P which is

$$\mathbf{A}^P = \mathbf{J}_P(\mathbf{q}, \dot{\mathbf{q}})\mathbf{q} + \mathbf{J}_I(\mathbf{q})\mathbf{q} \quad (28)$$

When the external force \mathbf{f}_P^x is zero the acceleration of P \mathbf{A}_P^0 is

$$\mathbf{A}_P^0 = \mathbf{J}_P(\mathbf{q}, \dot{\mathbf{q}})\mathbf{q} + \mathbf{J}_I(\mathbf{q})\mathbf{q}^0 \quad (29)$$

Subtracting (29) from (28) we get

$$(\mathbf{A}_I - \mathbf{A}_P^0) = \mathbf{J}_I(\mathbf{q})(\mathbf{q}^x - \mathbf{q}^0) \quad (30)$$

and combining (30) and (26) we find that

$$(\mathbf{A}_I^x - \mathbf{A}_P^0) = \mathbf{J}_P \mathbf{M}^{-1} \mathbf{J}_I^I \mathbf{f}_P^x \quad (31)$$

The above equation proves that the net acceleration effect of the force applied to P is proportional to the force's magnitude (Fact 2)

Suppose now that there is another external force \mathbf{f}_R^y acting on the figure at some point R . When both \mathbf{f}_P^x and \mathbf{f}_R^y are active the equation of motion of the figure will be

$$\mathbf{M}(\mathbf{q})\mathbf{q}^{(x,y)} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \mathbf{F} + \mathbf{J}_I^I(\mathbf{q})\mathbf{f}_P^x + \mathbf{J}_R^I(\mathbf{q})\mathbf{f}_R^y \quad (32)$$

Subtracting (25) from (32) we obtain

$$(\mathbf{q}^{(x,y)} - \mathbf{q}^0) = \mathbf{M}^{-1} \mathbf{J}_I^I \mathbf{f}_P^x + \mathbf{M}^{-1} \mathbf{J}_R^I \mathbf{f}_R^y \quad (33)$$

However from (26) we know that $\mathbf{M}^{-1} \mathbf{J}_I^I \mathbf{f}_P^x = (\mathbf{q}^x - \mathbf{q}^0)$ and $\mathbf{M}^{-1} \mathbf{J}_R^I \mathbf{f}_R^y = (\mathbf{q}^y - \mathbf{q}^0)$ therefore

$$(\mathbf{q}^{(x,y)} - \mathbf{q}^0) = (\mathbf{q}^x - \mathbf{q}^0) + (\mathbf{q}^y - \mathbf{q}^0), \quad (34)$$

which proves Fact 1 that the net effect on the acceleration of two forces acting simultaneously on a figure is equal to the sum of the net effects each one has on the same figure

We would like to stress once more that the two results we proved here hold only for the case when the time is considered frozen and they don't hold in general

2.9 Appendix 2 Featherstone's Recursive Dynamics

We first provide a short introduction to the spatial notation used by Featherstone's algorithm [19] and then explain how the algorithm works. We describe how complex joints can be modeled and finally provide the complete set of the recursive equations of motion for an arbitrary articulated figure.

2.9.1 Spatial Notation

Spatial vectors are 6 dimensional vectors which by combining linear and angular components in a single entity can be used to represent the physical quantities involved in rigid body motion. The power of spatial vectors lies in their compactness which allows rigid body systems to be described by a smaller number of equations relating fewer quantities than their traditional 3 dimensional counterparts allow. This section describes the relationship between 3 dimensional and spatial vectors, shows how physical quantities can be represented using the spatial notation and introduces the spatial coordinate transformations.

Two types of vectors appear in rigid body mechanics: *free* vectors and *line* vectors. Both free and line vectors have a particular magnitude and direction but a free vector can be positioned anywhere in space whereas a line vector, as the name suggests, is restricted to lie in a specific line. A line vector whose magnitude and direction are given by the vector \mathbf{a} and is restricted to lie on a line passing through the origin is represented in spatial notation as a 6 element column vector $\hat{\mathbf{a}}$.

$$\hat{\mathbf{a}} = [\mathbf{a}^T \ 0 \ 0 \ 0]^T$$

In general, if a line vector $\hat{\mathbf{a}}$ is restricted to lie in the line described by

$$\mathbf{r} \times \mathbf{a} = \mathbf{a}_O$$

where \mathbf{r} is a vector from the origin O to any point on the line, $\hat{\mathbf{a}}$ is written as

$$\hat{\mathbf{a}} = [\mathbf{a}^T \ \mathbf{a}_O^T]^T$$

A free vector whose magnitude and direction are given by the vector \mathbf{b} is written in spatial form as

$$\hat{\mathbf{b}} = [0^T \ \mathbf{b}^T]^T$$

Spatial vectors depend on the coordinate system they are represented in. A vector $\hat{\mathbf{a}}_O = [\mathbf{a}^T \ \mathbf{a}_O^T]^T$ represented in a coordinate system with origin O when viewed from a parallel system with origin P becomes $\hat{\mathbf{a}}_P = [\mathbf{a}^T \ \mathbf{a}_P^T]^T$ where

$$\mathbf{a}_P = \mathbf{a}_O + \overrightarrow{PO} \times \mathbf{a} \quad (35)$$

Equation 35 is called the *translation rule* for spatial vectors.

Any spatial vector can be considered as the sum of a line vector and a free vector. For a vector $\hat{\mathbf{a}} = [\mathbf{a}^T \ \mathbf{a}_O^T]^T$ the line and the free vector components are $[\mathbf{a}^T \ 0^T]^T$ and $[0^T \ \mathbf{a}_O^T]^T$ respectively. Based on this fact, one can define common physical quantities involved in rigid body dynamics as follows.

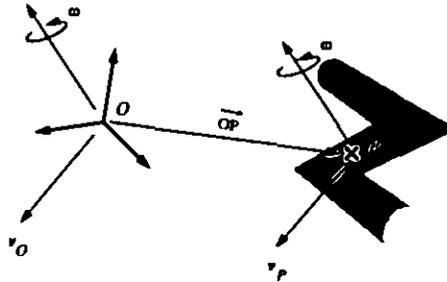


Figure 8 The velocity of an object with respect to an arbitrary coordinate system can be computed using the translation rule for spatial vectors

The instantaneous velocity of a rigid body has both an angular and a linear component. For any point P moving with the body the representation of the spatial velocity in a coordinate system with origin at P is

$$\hat{v}_P = [\omega^I \ v_P^I]^I$$

where v_P is the linear velocity of point P and ω is the angular velocity about an axis passing through P . To express the velocity of the body in a different coordinate system one can use the translation rule of equation 35 (figure 8). If the origin of the new coordinate system is O then the spatial velocity of the body is

$$\hat{v}_O = [\omega^I \ v_O^I]^I$$

where

$$v_O = v_P + \overrightarrow{OP} \times \omega$$

A force with magnitude and direction given by f acting along a line passing through a point P of the rigid body is written in spatial notation as

$$\hat{f} = [f^I \ \tau_O^I]^I$$

where $\tau_O = \overrightarrow{OP} \times f$ is the moment of the force about O

Spatial vectors can be used to define joints that connect the links of an articulated figure. In particular for a revolute joint (a joint that allows only rotation around an axis fixed in both links) if s is a unit vector along the direction of the joint axis and r is the position vector of any point on this axis then the spatial representation of the joint axis is

$$\hat{s} = [s^I \ s_O^I]^I \quad (36)$$

where $s_O = r \times s$. A prismatic joint (a joint that allows pure translation between two adjacent links in a particular direction) can be represented using a free vector since all prismatic joints with parallel axes have the same displacement effect. If s is a unit vector along the joint axis then the spatial representation of the prismatic joint axis is

$$\hat{s} = [0^I \ s^I]^I \quad (37)$$

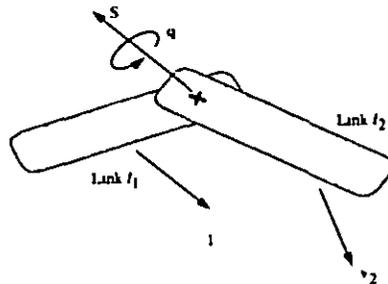


Figure 9 Propagating the link velocity through a joint. If link l_1 has velocity \hat{v}_1 and the joint velocity is q then link l_2 has velocity $v_2 = \hat{v}_1 + \hat{S}q$

One of the advantages of using spatial notation is that the treatment of joints is uniform regardless of their type. The same equations that apply for revolute joints can be used for prismatic joints. If $s = [s^T \ s_0^T]^T$ represents a joint connecting links l_1 and l_2 (figure 9) in an articulated figure then the spatial velocity of l_2 \hat{v}_2 is related to the spatial velocity of l_1 \hat{v}_1 through

$$\hat{v}_2 = \hat{v}_1 + \hat{S}q$$

where q is a scalar representing the joint velocity

The transformation of a spatial vector from one coordinate system to another is achieved through a 6×6 spatial transformation matrix. If \hat{a}_O is represented in a coordinate system with origin O then the same quantity represented in a parallel system with origin P \hat{a}_I could be expressed in matrix form as

$$\hat{a}_I = {}_I X_O \hat{a}_O$$

where ${}_I X_O$ is the spatial transformation matrix given by

$${}_I X_O = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{r} \times & \mathbf{1} \end{bmatrix}$$

In the above equation $\mathbf{1}$ is the 3×3 identity matrix, $\mathbf{0}$ is the 3×3 zero matrix, $\mathbf{r} = [r_x \ r_y \ r_z]$ is the vector \overrightarrow{PO} and $\mathbf{r} \times$ is the antisymmetric 3×3 matrix given by

$$\mathbf{r} \times = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}$$

If the transformation between coordinate systems involved only rotation instead of translation then the spatial transformation matrix has the form

$$\begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} \end{bmatrix}$$

where \mathbf{E} is the 3×3 orthogonal rotation matrix that transforms a 3 vector from one system to the other

Spatial transformations can be combined the same way Cartesian transformations are combined by multiplying the transformation matrices. A general spatial transformation that represents a shift of origin from O to P followed by a rotation around P is given by

$${}^I X_O = \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{E} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{r} \times & \mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{E} \mathbf{r} \times & \mathbf{E} \end{bmatrix}$$

where $\mathbf{r} = \overrightarrow{PO}$. The inverse transformation is

$${}^O X_P = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{r} \times^I & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{E}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{E}^{-1} \end{bmatrix} = \begin{bmatrix} \mathbf{E}^{-1} & \mathbf{0} \\ \mathbf{r} \times^I \mathbf{E}^{-1} & \mathbf{E}^{-1} \end{bmatrix}$$

In order to compute the scalar product between two spatial vectors the spatial transpose is defined as

$$\hat{\mathbf{a}}^S = \begin{bmatrix} \mathbf{a} \\ \mathbf{a}_O \end{bmatrix}^S = [\mathbf{a}_O^I \ \mathbf{a}^I]^I$$

The scalar product () is represented in terms of the spatial transpose as

$$\hat{\mathbf{a}} \hat{\mathbf{b}} = \hat{\mathbf{a}}^S \hat{\mathbf{b}}$$

The spatial inertia of a rigid body is a 6×6 matrix which when multiplied with the velocity of a rigid body yields the body's momentum. The spatial inertia of a body with mass m , center of mass at \mathbf{c} and rotational inertia \mathbf{I} about its center of mass is

$$\hat{\mathbf{I}} = \begin{bmatrix} m\mathbf{c} \times^I & m\mathbf{1} \\ \mathbf{I} + \mathbf{c} \times m\mathbf{c} \times^I & \mathbf{c} \times m \end{bmatrix} = \begin{bmatrix} \mathbf{H}^I & \mathbf{M} \\ \mathbf{I} & \mathbf{H} \end{bmatrix}$$

2.9.2 The Articulated Figure Representation

An articulated figure consists of a number of links connected by joints (figure 10). Featherstone's algorithm deals with arbitrary tree-like structured articulated figures or in other words articulated figures that do not contain closed kinematic loops. If we pick one of the links of the figure and call it the *root link* then a structural hierarchy can be defined on the figure. Each link except from the root, has a *parent link* through which it is connected to the figure. A link has only one parent but can have multiple children.

Joints connecting the links can have up to three translational and three rotational degrees of freedom (DOFs). However in the spatial notation a joint is represented with one spatial vector corresponding to the joint's axis. Therefore a spatial joint is allowed only one DOF. We can overcome this limitation by introducing dummy links between the actual figure links connected together with single DOF spatial joints. To represent a k DOF joint, $(k - 1)$ dummy links are needed (figure 11). Dummy links are massless and do not have any dimensions but they are treated just like the rest of the figure's links.

To deal with the figure's links consistently we assign a distinct number to each one of them. If the figure has n links (counting the dummy links too) then we assign the number 0 to the root link and a

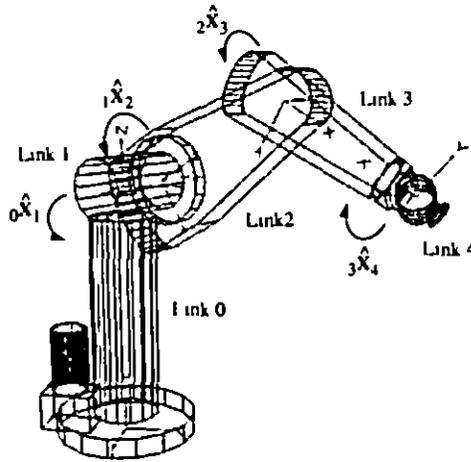


Figure 10 An example of an articulated figure with 5 links. Each link has a local coordinate system attached to it. The transformations from the link to the parent coordinate system are shown. In this example we assume that all the joints of the figure have 1 DOF.

number from 1 to $n - 1$ to the rest of the links in such a way as to make sure that a link always gets a higher number than its parent. For each link l we define p_l to be l 's parent link and c_l to be a set (possibly empty) of l 's children. Furthermore, we attach a local coordinate system to each link. Having introduced the local coordinate systems, we can define the transformation from link l to its parent to be ${}_{p_l}X_l$. Finally, we call I_l the spatial inertia of link l and \hat{S}_l axis of the one DOF joint that connects l to its parent as defined in equations 36 and 37.

2.9.3 The Articulated-Body Dynamics Algorithm

The basic idea of Featherstone's Articulated Body Method is to regard the articulated figure as consisting of one base member (whose motion is known) connected through a joint to a moving link (which in reality represents the rest of the figure). This reduces the original problem to determining the motion of a one joint figure which is easily solved given the apparent inertia of the moving link. Once we determine the acceleration of the joint, we can add the link connected with that joint to the base and continue to obtain the acceleration of the next joint (figure 12). To carry out the algorithm we therefore need to be able to compute each time the apparent inertia of the moving link and to solve the dynamics of a one joint figure with a moving base.

The apparent inertia of the link is called the *articulated body inertia*, denoted by \hat{I}^A and is defined as the relationship between a force f applied to the link from the base and the acceleration of the link \hat{a}_l .

$$\hat{f} = \hat{I}^A \hat{a}_l + p \quad (38)$$

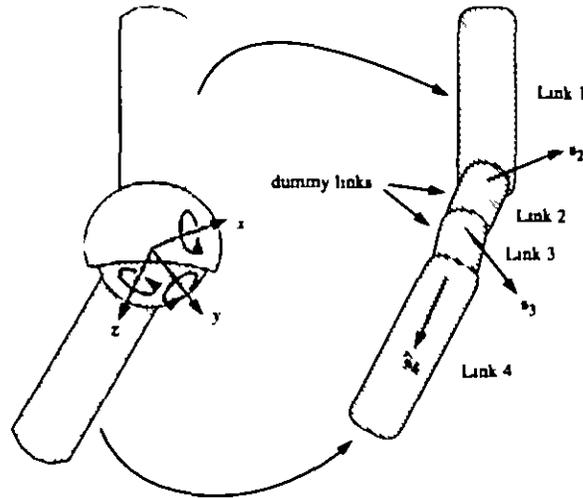


Figure 11 The spherical joint is an example of a joint with multiple DOFs. The three rotational DOFs of the joint can be represented in spatial form by introducing two dummy links to the articulated figure. Each link l is connected to its parent via a one DOF joint with axis \hat{s}_l . For drawing purposes we show the dummy links as having some length. In practice there is no displacement between \hat{s}_2 and \hat{s}_1 .

In the above equation \hat{p} is called the bias force and is the force which must be applied to the link from the base in order to give to the link zero acceleration.

We will now show how the one joint figure problem is solved if the value of $\hat{\mathbf{I}}^{-1}$ is known. Consider a one joint figure with a base moving with velocity \hat{v}_b and acceleration \hat{a}_b . If the link has inertia $\hat{\mathbf{I}}$ and the joint has an axis \hat{s} velocity q and acceleration \dot{q} then the velocity and acceleration of the link can be written as

$$\hat{v}_l = \hat{v}_b + \hat{s}q \quad (39)$$

and

$$\hat{a}_l = \hat{a}_b + \hat{v}_b \times \hat{s}q + \hat{s}\dot{q} \quad (40)$$

In equation 40 the term $\hat{v}_b \times \hat{s}q$ is the Coriolis acceleration term which is due to the fact that the base is moving. The equation of motion for the link can be obtained by differentiating Newton's equation. Since the net force acting on the link is equal to the link's rate of change of momentum we have

$$\begin{aligned} \hat{f} &= \frac{d}{dt}(\hat{\mathbf{I}}\hat{v}_l) \\ &= \hat{\mathbf{I}}\hat{a}_l + \hat{v}_l \times \hat{\mathbf{I}}\hat{v}_l \end{aligned} \quad (41)$$

If we ignore gravity \hat{f} can only be applied to the link through the joint. Its component along the joint axis is

$$Q = \hat{s}^S \hat{f} \quad (42)$$

Since force cannot be transferred from the base to the link along the joint axis, the component of the force along the joint axis that reaches the link can only be due to the internal joint force/torque applied

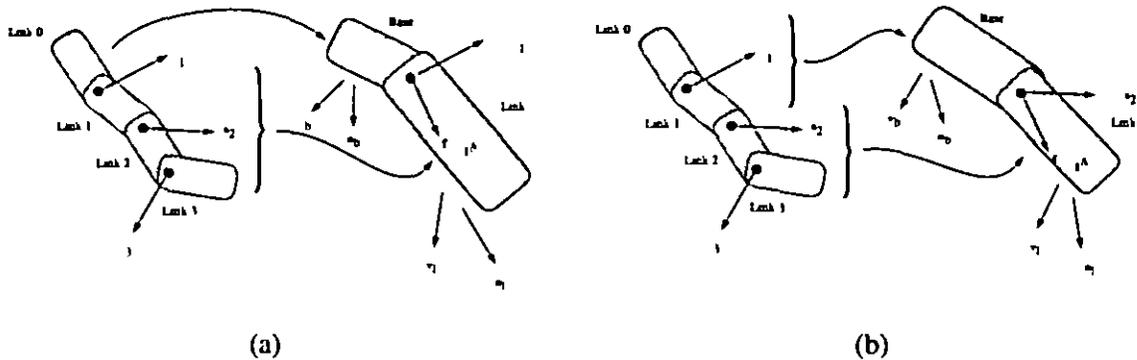


Figure 12 Computing the articulated figure dynamics. The figure links are divided into two sets: the Base contains those links whose motion is known and the Link that contains the links whose motion we want to determine. (a) In the first iteration of the dynamics algorithm, the base consists only of link 0 and the Link of the rest of the links. Once the joint acceleration of link 1, q_1 , is computed, link 1 becomes a member of the base and the procedure is repeated as shown in (b). In this example, the acceleration of all the links can be computed after 3 repetitions of the same procedure.

from the joint's actuator (if there is one) and is therefore a known quantity. If the joint is not actuated then $Q = 0$. Substituting equations 40 and 41 into equation 42, we can obtain an expression for q in terms of Q :

$$q = \frac{Q - \hat{s}^s (\hat{\mathbf{I}}(\hat{\mathbf{a}}_b + \hat{\mathbf{v}}_b \times \hat{\mathbf{s}}q) + \hat{\mathbf{v}}_l \times \hat{\mathbf{I}}\hat{\mathbf{v}}_l)}{\hat{s}^s \hat{\mathbf{I}}\hat{\mathbf{s}}} \quad (43)$$

We now replace the single rigid body of the link by the part of the articulated figure it really represents and we use equation 38 instead of 41. The above expression for q now becomes

$$q = \frac{Q - \hat{s}^s (\hat{\mathbf{I}}^{-1}(\hat{\mathbf{a}}_b + \hat{\mathbf{v}}_b \times \hat{\mathbf{s}}q) + \hat{\mathbf{p}})}{\hat{s}^s \hat{\mathbf{I}}^A \hat{\mathbf{s}}} \quad (44)$$

This is the equation of motion for one joint of the articulated figure. We start by using the root link as the base and every time a new joint acceleration is computed, we include the corresponding link in the base. By repeating the process for all the joints of the figure (n times), we can compute the acceleration of all the joints.

Featherstone in his book [19] presents a detailed derivation of the articulated body inertias. Here we will only repeat the results by providing the final form of the equations of motion of the whole articulated figure.

2.9.4 Final Form of the Motion Equations

In a forward pass, for each link l starting from 0 and going to n , we compute

$$\hat{\mathbf{v}}_l = {}_l\mathbf{X}_{p_l} \hat{\mathbf{v}}_{p_l} + \hat{\mathbf{s}}_l q_l \quad (45)$$

$$\hat{c}_l = \hat{v}_l \times \hat{s}_l q_l \quad (46)$$

$$\hat{p}_l^v = \hat{v}_l \times \hat{I}_l \hat{v}_l - \hat{f}_l^f \quad (47)$$

Equation 45 should not be used for the root link. The value of \hat{v}_0 is 0 at the beginning of the simulation and as time advances it can be computed through integration of the acceleration \hat{a}_0 . In equation 47 \hat{f}_l^f represents the external forces acting on the link l expressed in the local coordinate system of l .

In a backward pass from the leaves of the structure towards the root we compute

$$\hat{I}_l^A = I_l + \sum_{i \in c_l} {}_l X_i \left(\hat{I}_i^A - \frac{\hat{h}_i \hat{h}_i^S}{d_i} \right) {}_l X_i \quad (48)$$

$$\hat{h}_l = \hat{I}_l^A \hat{s}_l \quad (49)$$

$$d_l = \hat{s}_l^S \hat{h}_l \quad (50)$$

$$p_l = \hat{p}_l^v + \sum_{i \in c_l} {}_l X_i \left(\hat{p}_i + \hat{I}_i^A c_i + \frac{u_i}{d_i} \hat{h}_i \right) {}_l X_i \quad (51)$$

$$u_l = Q_l - \hat{h}_l^S \hat{c}_l - \hat{s}_l^S \hat{p}_l \quad (52)$$

For links that do not have children equation 51 is replaced by $\hat{p}_l = \hat{p}_l^v$ and equation 48 is replaced by $\hat{I}_l^A = \hat{I}_l$.

Finally in a forward pass

$$\hat{a}_l = {}_l X_{p_l} \hat{a}_{p_l} + \hat{c}_l + \hat{s}_l q_l \quad (53)$$

$$q_l = \frac{u_l - \hat{h}_l^S {}_l X_{p_l} \hat{a}_{p_l}}{d_l} \quad (54)$$

For the root link equation 53 is replaced by

$$\hat{a}_0 = -(\hat{I}_0^A)^{-1} \hat{p}_0 \quad (55)$$

2.10 Summary

We have presented a physics based system for animating articulated figures. We showed how one of the most efficient forward dynamics simulation algorithms for articulated figures, Featherstone's Articulated Body method, can be extended to handle arbitrary acceleration constraints. Our goal was to present an easy to implement and use constraint force evaluation algorithm that effects as little as possible the overall performance of the dynamics simulation. We showed how the acceleration constraints are used in our system to simulate more complex effects such as impacts and contacts and model joint limits and closed kinematic loops in the figure. Finally we demonstrated how by using constraints for trajectory following one can robustly control the motion of articulated figures. We believe that the animation system we described can be an extremely powerful tool in the hands of a computer animator.

3 Capturing Human Motion and Integration with Recursive Dynamics

Ioannis Kakadiaris¹ and Dimitris Metaxas

¹Dept. of Computer Science, Univ. of Houston, Houston, TX 77204-3475. Email: ioannisk@uh.edu

On one hand, the capture and automatic generation of realistic movement is a very important yet open problem in computer graphics and animation. On the other hand, computer vision techniques are playing an increasingly important role in computer graphics applications requiring enhanced realism [4-64]. This synergy between computer vision and computer graphics is of paramount importance in virtual reality applications where the use of complex models such as humans or other articulated objects and the modeling of their motions is often necessary [5-16]. In such cases, even the most skilled modelers and animators are not able to accurately reproduce the respective shapes and motions. If synthesized motion is to be compelling, we must create actors for computer animations and virtual environments that appear realistic when they move. The realistic animation of virtual humans requires that 1) the figure must be anthropometrically correct and resembles the participant, and 2) the kinematics and the dynamics of the figure must be accurate and physically correct. Other applications, such as vision-based computer interfaces [39-53], ergonomics, anthropometry, and human factors design, require additional analysis of the data to facilitate certain tasks or activities. An example of such an analysis is the performance measurement of both athletes as well as patients with psychomotor disabilities and the rapid prototyping of rehabilitation aids [66].

Recently, we proposed the following approach in order to capture the motion of an unencumbered subject and to overcome the problems due to variations among subjects in the shape of their body. First, automatically acquire a three-dimensional model of the subject using vision techniques [34]. Then, employ this model to track the unconstrained movement of a subject's body parts by using multiple cameras and by actively selecting the subset of the cameras that provide the most informative views [35]. Since the system is non-invasive, the subject can move freely without interference from the system and without markers annoying him/her.

In the following sections, we will present in detail the integration of these vision techniques into a vision-based animation system. Our system consists of a human motion analysis and a human motion playback parts. The analysis part is based on the spatio-temporal analysis of the subject's silhouette from image sequences acquired simultaneously from multiple cameras [33]. The motion playback part couples the estimated motion parameters with a customized physics-based graphical model of the participant, resulting in human motion animation. The system is currently passive, that is, the observations drive the model. Since our motion capture technique is model-based, the mapping between the states of the system and the image features makes registration possible. However, the state space must be observable. During the *selection* phase of the motion estimation part, we derive for each part a subset of the cameras that provide the most informative views for tracking. This *active* and time-varying camera selection, which allows for robustness and speed of processing, is based on the visibility of a part and the observability of its predicted motion from a certain camera. We offer a detailed analysis of the singularities that might arise during tracking humans using multiple cameras and we provide an analysis of the criteria that allow the ordering of the cameras that provide the most information for tracking.

We used our system to capture the motion of the upper body extremities of several subjects performing complex three dimensional motions in the presence of significant occlusion. We achieved accurate tracking results and could successfully animate a customized model of the subject. In addition we will demonstrate two case studies of possible applications of vision based animation namely performance measurement of people with disabilities for rapid prototyping and vision based trajectory input for dynamic simulators. This new system advances the state of the art in vision based animation because 1) our method is based on the use of occluding contours and it obviates the need for markers or special equipment 2) the animated graphical model is anthropometrically correct and resembles the subject 3) the selection of the subset of cameras that provide the most informative views is accomplished in an active and time varying fashion 4) since we are using multiple cameras we can mitigate the difficulties arising due to occlusion among body parts and body movements that are ambiguous from one view can be disambiguated from another view and 5) the motion capture is based on the whole apparent contour than just a few points.

The work we will present is organized as follows. Section 3.1 presents a brief survey of the current state of the art in human motion capture. Section 3.2 presents the recipe for the estimation part of our system along with a detailed analysis of the tracking singularities. Section 3.3 presents the motion playback part. Section 3.4 details our experiments and the analysis of the motion capture accuracy. Section 3.5 discusses the results.

3.1 Related Research

Currently motion tracking is performed using mechanical, electro-magnetic, and image based techniques. Image-based systems can be lumped into three broad categories: techniques that use active markers, techniques that use passive markers, and techniques that do not use markers. Since placing markers to the subjects is cumbersome and it alters their natural motion, a non-intrusive sensory method based on vision is preferable. Indeed, there is a need for accurate measurement of the three dimensional motion of moving body parts without interfering with the dynamics of the moving bodies. Since our method is image based we review only image based marker free approaches. For a comprehensive review of human tracking methods the reader is referred to [20-33]. Previous work on human motion capture can be categorized in bottom-up methods [30] (where no prior knowledge about the shape model of the human is employed) and top-down model based methods which employ a model of the human body or employ assumptions on the kinds of motions they can analyze [11, 49-62]. Most of the top-down techniques use approximate rigid models of the human body (e.g. stick figures [17, 37, 27, 11], overlapping spheres [50], cylinders [2, 61], elliptical cylinders [29, 62, 41], right-circular cones [25], or superquadrics [21, 42, 52]) which cannot easily adapt to different body sizes. Some approaches perform tracking using edges [21, 35, 25, 29, 60] or using information related to regions [54, 61, 73, 13] or both [67]. Alternatively, 2D kinematic models have been used to track end-effectors using color information [71] and leg configurations [12].

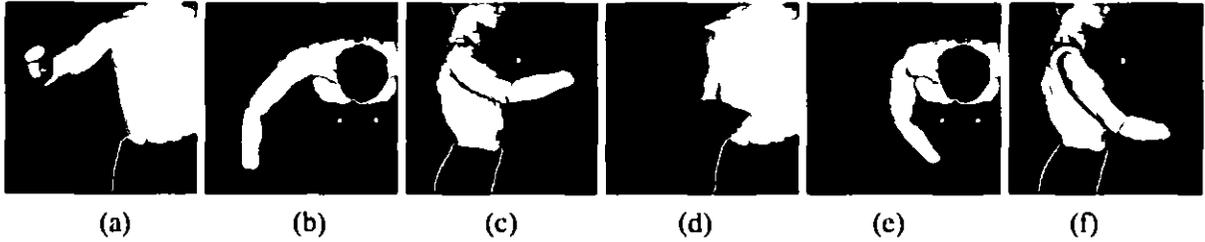


Figure 13 (a c) Starting and (d f) ending position of the right upper arm and forearm of a graphical model

3.2 Motion Capture

To capture in a non intrusive fast and robust fashion the geometric and kinematic parameters of humans we have constructed an experimental testbed (3D studio). The imaging hardware of the 3D studio consists of three calibrated grayscale cameras placed in mutually orthogonal configuration. In particular the input to the analysis part is a sequence of grayscale images taken from these cameras. The output is the three dimensional positions and orientations of the subject's body parts at each time instant from which the trajectories velocities and accelerations can be computed. The first phase of our system is the model building phase. That is we first extract a shape model of the subject's parts which will allow prediction of the occlusions among body parts (see [?]). The input to our system are the subject's occluding contours. The reconstruction algorithm then synthesizes the contours from the three image sequences to build the shape representation of each body part. For the experiments described the subject is asked to perform a set of motions to acquire the anthropometric dimensions of his/her upper and lower arms.

The above derived shape of the subject's body parts is used in the motion capture phase. We will now present in detail the steps of the motion capture process that take place for every frame of the image sequence. In the pseudo code below $startI$ and $endI$ are the starting and ending times respectively $\mathbf{u} = (\mathbf{q}^T \ \dot{\mathbf{q}}^T)^T$ where \mathbf{q} is the vector of generalized coordinates that describe the articulated model M of the subject with n parts. The elements of \mathbf{q} are the positions and orientations of the models of each modeled body part of the subject's body. The vector \mathbf{q} along with the vector $\dot{\mathbf{q}}$ completely describe the state of articulated model of the subject. The matrix \mathbf{P}_{start} represents the uncertainty of the state. The motion estimation process proceeds as follows:

```

MotionEstimation( $startI$   $endI$   $M$   $\mathbf{u}_{start}$   $\mathbf{P}_{start}$ )
 $t_k \leftarrow startI$ 
 $\mathbf{u}(t_k|t_k) \leftarrow \mathbf{u}_{start}$ 
 $\mathbf{P}(t_k|t_k) \leftarrow \mathbf{P}_{start}$ 
while ( $t_k < endI$ )
   $\{\mathbf{u}(t_{k+1}|t_k) \ \mathbf{P}(t_{k+1}|t_k)\} \leftarrow \mathbf{Predict}(\mathbf{u}(t_k|t_k) \ \mathbf{P}(t_k|t_k))$ 
  SynthesizeAppearance( $\mathbf{u}(t_{k+1}|t_k)$ )
   $\{BC_1 \ \dots \ BC_n\} \leftarrow \mathbf{SelectBestCamera}(\hat{\mathbf{u}}(t_{k+1}|t_k) \ M)$ 

```

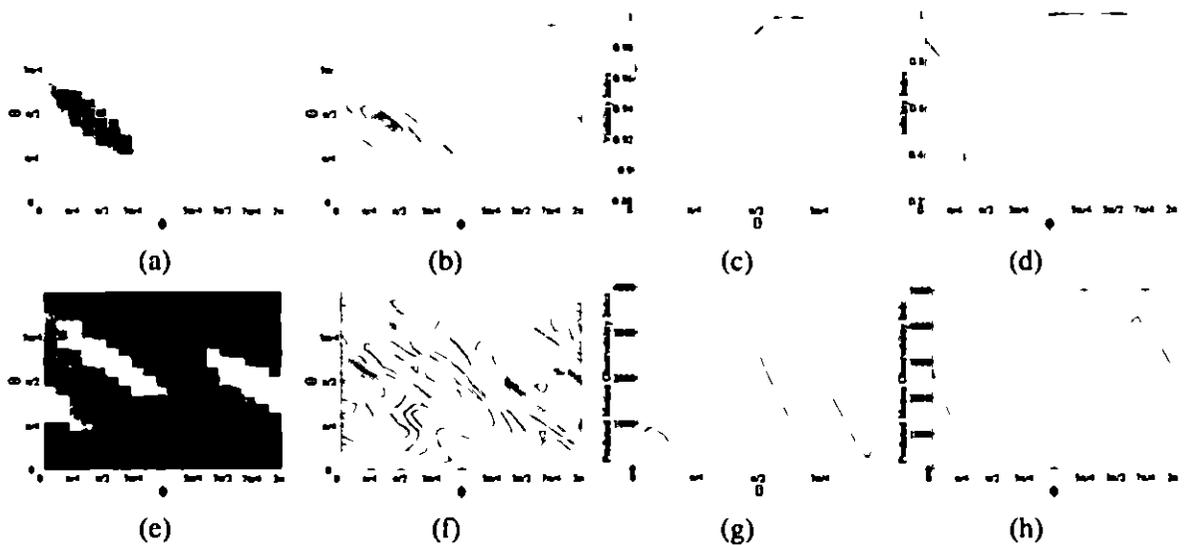


Figure 14 (a, e) Intensity maps and (b, f) iso contour plots of the variation of the visibility index and the predicted motion observability index for the right upper arm of the graphical model respectively (c, g) Variation of the indexes for the right upper arm along θ for $\phi = 0$ (d, h) Variation of the indexes for the right upper arm along ϕ for $\theta = 0$

```

MatchAndMeasure(u(tk+1|tk) {BC1 BCn})
StateUpdate(u(tk+1|tk) P(tk|tk) P(tk+1|tk))
tk ← tk + 1

```

The procedure *Predict* takes into account states up to time t_k to make a prediction for the state of the model at time t_{k+1} . The procedure *SynthesizeAppearance* synthesizes the appearance of the graphical model for each camera while the procedure *MatchAndMeasure* establishes the discrepancy between the predicted and the actual appearance. Then a new state for the model is estimated by the procedure *StateUpdate* so as to minimize these discrepancies. The process is repeated again for the following frames in the image sequence. Since we are using multiple cameras one of the questions we are called upon to answer is how many views should we employ for tracking. Should we process the occluding contours from all the cameras or from only one? Therefore we have added to the algorithm the *SelectBestCamera* procedure that (for each body part) derives the camera that provides the most informative view for tracking [35]. In the following we will present a detailed account for choosing the criteria.

3.2.1 Prediction Phase

We incorporate the dynamics of the subject's model into a Kalman filter formulation [22, 35]. The Kalman filter formulation is used because 1) it provides the optimal linear estimate for dynamic systems 2) it is recursive and therefore computationally efficient and 3) it is based on physical dynamics.

which allows for predictive estimation. Let $\mathbf{u}(t_2|t_1)$ denote the minimum mean square error estimate of $\hat{\mathbf{u}}(t_2)$ given the measurements up to t_1 ($t_1 \leq t_2$) and $\mathbf{P}(t_2|t_1)$ the associated error covariance matrix. Then the prediction component of the Kalman Filter can be described using the following pseudo code

```

Predict( $\mathbf{u}(t_k|t_k)$   $\mathbf{P}(t_k|t_k)$ )
   $\mathbf{u}(t_{k+1}|t_k) \leftarrow \text{PredictState}(\mathbf{u}(t_k|t_k))$ 
   $\mathbf{L}(t_{k+1}) \mathbf{L}_p(t_{k+1}) \leftarrow \text{EvaluateJacobians}(\mathbf{u}(t_{k+1}|t_k))$ 
   $\mathbf{P}(t_{k+1}|t_k) \leftarrow \text{PredictCovariance}(\mathbf{P}(t_k|t_k) \mathbf{Q}(t_k))$ 

```

The entries of matrix $\mathbf{L}_p(t_{k+1})$ consist of evaluations of the Jacobian matrix \mathbf{L}_p at the various locations of the projected model points [35]. Using the above extended Kalman filter we can predict at every step the expected location of the model point positions in the next image frame based on the magnitude of the estimated parameter derivatives \mathbf{q} .

3.2.2 Synthesizing the Appearance of the Model

Based on the predicted position of the shape model the algorithm synthesizes the appearance of the model to the different cameras. For every model the projection of the vertices of the model to the image plane of each camera is computed.

3.2.3 Selection Phase

Our goal is to track (using the recovered shape model for the subject) the three dimensional position and orientation of the subject's body parts. Using portions of the contours from all the cameras simultaneously to compute forces on a particular part can result into an incorrect solution due to incorrect association of contour points to model points. Therefore at every step of the computation we decide which cameras to use by employing the procedure below.

```

SelectBestCamera( $\mathbf{u}(t_{k+1}|t_k)$   $\mathbf{M}$ )
  for  $i = 1$  to  $n$ 
    for  $j = 1$  to  $k$ 
       $\mathcal{VI}_{i,j} \leftarrow \text{ComputePartVisibility}(\mathbf{u}(t_{k+1}|t_k) \mathbf{M})$ 
    for  $i = 1$  to  $n$ 
      if ( $i > \mathcal{VI}_{thr(i)}$ )
        then
           $\mathcal{OI}_{i,j} \leftarrow \text{ComputeProjMotObservability}(\mathbf{u}(t_{k+1}|t_k) \mathbf{M})$ 
        else
           $\mathcal{OI}_{i,j} \leftarrow 0$ 
    for  $i = 1$  to  $n$ 
       $BC_i \leftarrow \text{argmax}_j \mathcal{OI}_{i,j}$ 

```

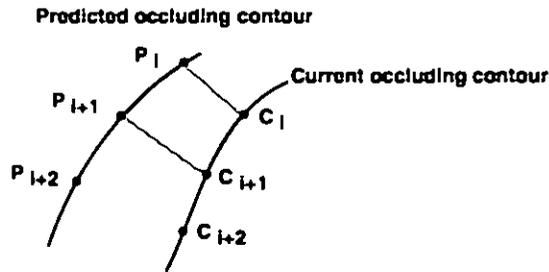


Figure 15 Notation pertaining to the computation of the observability index

The procedure *ComputePartVisibility* computes the visibility of a subject's body part from a particular camera and the procedure *ComputeProjMoiObservability* computes the observability of its motion w.r.t the particular camera. For a part i and a camera C_k , $k \in \{1, 2, 3\}$ we define a visibility index $\mathcal{V}I_{i,j}$ and an observability index $\mathcal{O}I_{i,j}$.

Part Visibility Criterion We define as visibility index $\mathcal{V}I_{i,j}$ the ratio of the area of the visible projection of the part i to the image plane of camera C_k to its projected area when we do not take into account the occlusions. The visibility index is a function of the shape and pose of the object, the pose of the camera and the pose of other objects in the scene. To motivate the use of this index we have performed a simulation whose purpose was demonstrate the change to the visibility of an object (due to occlusions) as a camera moves around it in a unit sphere (the position of the camera is determined by the spherical coordinates ϕ and θ). For example imagine a sphere enclosing the object and a camera positioned at any point of this sphere looking at the object. Using the *Transom Jack*® software we have generated synthetic images of an arm at a specific pose for different positions of camera around it. Figs 14(a-d) depict the change in visibility index of the upper arm. In particular Fig 14(a) depicts as an intensity value the value of the visibility index as the camera sweeps the sphere around the graphical model of the arm. The white areas correspond to positions of the camera from which the arm is not occluded by other objects present in the scene. Fig 14(c) depicts the variation of the visibility index as the camera moves along the longitude of the sphere (at latitude 0) and Fig 14(d) depicts the variation of the visibility index as the camera moves along the latitude of the sphere (for longitude 0). It is hardly a surprise that there is a wide variation in the visibility index. Our motivation is just to reinforce that there are zones of high visibility and zones of low visibility. Therefore if one has the opportunity to choose a camera from a set of cameras it is relevant to chose the cameras that meet the visibility criteria.

Observability Criterion The second criterion for view selection refers to the observability of motion. Let's assume that we observe the movement of the upper arm and the forearm of a graphical model using three cameras. The same motion in 3D induces different change in the occluding contours from the three cameras. This change is function of the shape and pose of the object, its motion, occlusions and the pose of the observer. Our objective is to select the camera in which the largest change occurs since our algorithm is based on occluding contour information.

Once a part is considered visible from a particular camera, we further check if the camera's view point is degenerate given the predicted motion (based on the Kalman Filter) of the part. For every node

C_i , at the current occluding contour of the model we determine the nearest point P_i to the predicted occluding contour and we compute the area of the polygon with vertices C_i , C_{i+1} , P_i and P_{i+1} (Fig 15). The sum of these areas is the observability index and summarizes the changes at the apparent contour of an object for a given motion [35].

To motivate the use of the observability index we recorded the predicted motion observability index as the graphical model moved from the starting to the ending position (Fig 13). Fig 14(e) depicts as an intensity value the value of the observability index as a function of the camera position. The white areas correspond to positions of the camera to which the specific motion for the arm induces the largest change in image coordinates. In addition, this map reflects the axes of symmetry of the object. It gives the axes around which when the object moves its apparent contour does not change. Fig 14(g) depicts the variation of the observability index as the camera moves along the longitude of a sphere around the arm. Fig 14(h) depicts the variation of the observability index as the camera moves along the latitude of a sphere around the arm.

3.2.4 Matching and Measurement Phase

Having selected the set of cameras that provide the most information in order to provide constraints to the estimation of the rotational and translational motion of each of the parts we employ information from the occluding contours.

Specifically, assuming that the initial pose of the parts is known (we have developed a graphical tool that allows easy overlay of models onto image sequences for initialization purposes) for every occluding contour we employ the algorithm below.

```

MatchAndMeasure( $u(t_{k+1}|t_k)$ ,  $\{BC_1, \dots, BC_n\}$ )
  for  $j$  in  $\{BC_1, \dots, BC_n\}$ 
    Fit2DDeformableModel( $j$ )
    MatchPoints( $j$ ,  $u(t_{k+1}|t_k)$ )
    MeasureDifference( $u(t_{k+1}|t_k)$ )

```

The procedure *Fit2DDeformableModel* fits a two dimensional deformable model to the occluding contour in camera j to obtain a smooth differentiable model of the curve [42]. The procedure *MatchPoints* computes the correspondence between nodes on the model and points on the occluding contour. The procedure *MeasureDifference* measures the inconsistencies between the synthesized appearance of the model and the actual one. In physics-based tracking terms, it measures the forces that the image points apply to the model. After we compute the force assignments we estimate the new pose parameters of the part based on the extended Kalman filter [33].

State Update The estimation of the motion parameters is achieved by numerically integrating through time the Lagrange equations of motion taking into account the simulated external forces and the constraint forces.

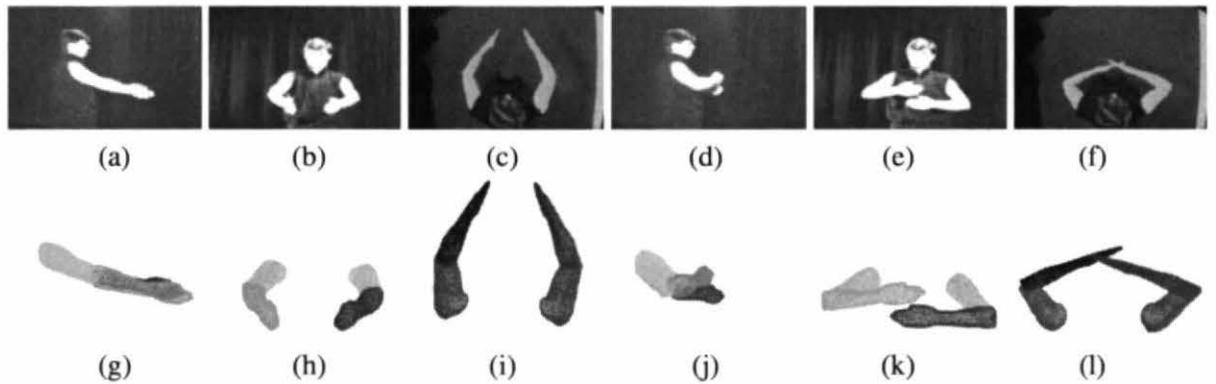


Figure 16: (a-c, d-f) Sample input images from the three cameras for frames 1 and 16 in the motion sequence, and (g-i, j-l) tracking results for frames 1 and 16 in the motion sequence.

3.3 Motion Playback

The above vision-based analysis gives the ability to produce a customized version of each individual's shape and motion which can be then played-back in any standard human animation system. For our purposes we have used the *Transom Jack*[®] animation system. The recovered shape of the parts is used to replace the standard shape of the parts used in this animation system. Therefore, the animated graphical model will resemble the subject. In particular, since three views are not sufficient to represent accurately a human's head and face, the head is captured using our Cyberware scanner and the shape of the animated body parts is obtained using the method described in [34]. The controlled degrees of freedom that we employ in the experiments section are the following: the neck, and (right and left) shoulder, and the elbow. The captured shape and motion of each human part is represented as a rigid body which rotates and translates over time. In addition the parts are connected with point-to-point constraints whose location is also defined during the vision-based analysis part.

3.4 Results

We carried out several experiments to assess the robustness and accuracy of our tracking scheme. The first experiment illustrates the robustness to occlusions. The process of tracking the motion of the subject's arms consists of two phases. The first phase is the model building phase. Notice that since the subject did not flex his wrists, the estimated models for the forearms include the wrists. In the second phase, we asked the subject to move his arms in 3D. Figs. 16(a-c, d-f) show sample input images from the three cameras. Figs. 16(g-i, j-l) show the estimated 3D pose of the subject's parts of the arm. Fig. 17(a) shows four views for three frames of an animation. The animation was created by applying the estimated motion parameters to a graphical model of the subject whose shape has been customized based on the shape of the subject's body parts (as extracted from the image sequences). In addition, using the recovered parameters of motion we can animate the vision-based extracted graphical model of the subject placed at a different synthetic environment (Fig. 17(b)).

The second experiment demonstrates the performance of our algorithm. The experimental proto-

col was the following: a subject was asked to slide the tip of his right hand along a circular groove $5\text{mm} \pm 0.5\text{mm}$ wide engraved in a glass plate, thus following a circular trajectory of radius $114.3\text{mm} \pm 0.5\text{mm}$ (Figs 18(a, c, i, c, g, k)). After tracking (Figs 18(b, f, j, d, h, l)) and trajectory reconstruction, an error analysis was performed which is summarized in Fig 18(p). Even by taking into account human imprecision while following the groove, the mean error between the ideal and the recovered trajectory of the finger tip is about 1cm for observation distance of 2.5 m. Thus the error is 0.4%. Figs 18(m, o) depicts the recovered path from several views. In Fig 18(n) all the axes have been drawn at the same scale. However, in Fig 18(m) the Z axis is drawn on smaller scale to depict the variation of the path. Fig 18(o) depicts the path in the transverse plane.

Case Studies UPGRADE is a research project being conducted at the GRASP Lab in collaboration with the A I duPont Institute. The goal of the project is to merge several state of the art technologies for the rapid prototyping of rehabilitation aids *customized* for a specific physically-challenged user. The emphasis is placed on providing accurate physical measurements to simulation tools, which in turn provide relevant *customizing* parameters to the design and manufacturing processes for rehabilitation aids.

We have chosen to investigate how a vision based motion capture and animation system can be used for the acquisition of kinematic and geometric data of motion impaired users, in order to provide measurements to rehabilitation device designers. We considered as a testbed the device showed in Fig 18(s) where the three dimensional head and neck motions of a quadriplegic patient can control a feeding utensil via the extension and retraction of cables (the cables and the attachment to the patient's head are not shown for simplicity). In this case, our goal is to measure how and to what extent a given patient can perform the motion controlling the feeding apparatus. The subject is being observed by the set of three CCD cameras of the 3D studio. A deformable model of the subject head is automatically fitted to the first image of the sequence and then tracked. Fig 18(r) shows the trajectory followed by a point on the chin of our test subject. Note that the motion is much more complex than the simple arc one could expect. The kinematic data thus acquired is used as input to a virtual prototyping module, where a tentative design for the feeding device can be simulated and refined. In particular, an optimization procedure was developed to compute the optimal mechanism dimensions and the motion coupling required to generate a desired spoon motion based on the recovered user specific 3D head trajectory (see Fig 18(q)) [36]. Eventually, a parameterized CAD model of the mechanism can be adjusted to the patient's specific needs and manufactured accordingly. It is crucial to note that our framework is flexible and equally allows for other type of input motions (e.g. arm, leg, ...) to be tracked and used for other instances of customized manufacturing, such as different types of rehabilitation aids or more generally other man/machine interfaces.

As a second case study, we employed the results of human motion analysis as trajectory input for a dynamic simulator. Figs 18(t, w) show a puppet that dynamically follows the trajectory (as it was estimated using our motion analysis system) that the tip of the hand follows.

3.5 Summary

Computer animations and interactive virtual environments involving humans require the generation of realistic human motion. We maintain that probably the best and most promising approach to this prob-

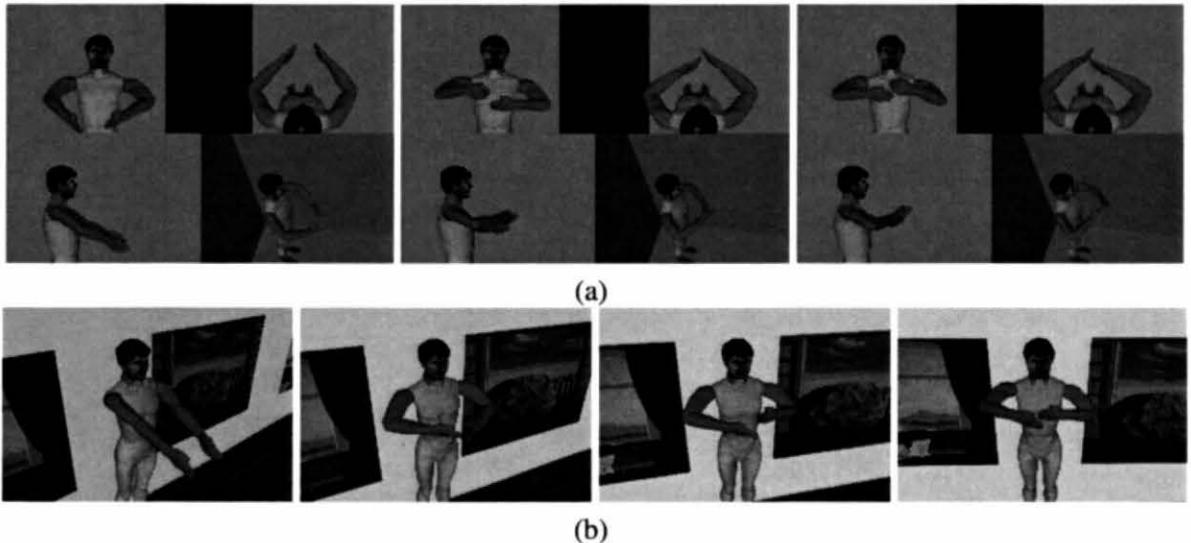


Figure 17: The estimated motion parameters for the motion of the subject’s arms have been applied to a graphical model of the subject. (a) Three frames from four different views. (b) The customized graphical model of the subject in a different virtual environment.

lem is a hybrid approach. Based on this approach, parts of the virtual actor’s motion are generated using motion capture data and others are driven by simulation and control algorithms [28]. We presented in detail the integration of the vision techniques into a vision-based animation system. We offered a detailed analysis of the singularities that might arise during tracking humans using multiple cameras and we provided an analysis of the criteria that allow the ordering of the cameras that provide the most information for tracking. In addition, we presented a detailed performance analysis of the motion capture technique for the case of tracking upper body extremities. Our experiments have demonstrated that we can indeed accurately estimate the motion parameters of a moving subject. In addition, we have demonstrated the ability to create animation sequences through the vision-based analysis of the video input. Our method can be used to make libraries of reusable motion. For example, techniques such as the ones presented in [15, 70, 23] could be used to adapt previously acquired motions to new situations and new characters or serve as databases for the “on-the-fly” assembly of animation in interactive systems.

4 Controlling a Dynamic System with Open and Closed Loops: Application to Ladder Climbing

Janzen Lo, Dimitris Metaxas and Norman I. Badler

Techniques for dynamically simulating humans and their behaviors in virtual worlds are becoming increasingly important in mechanical engineering, medical, military and computer graphics applications.

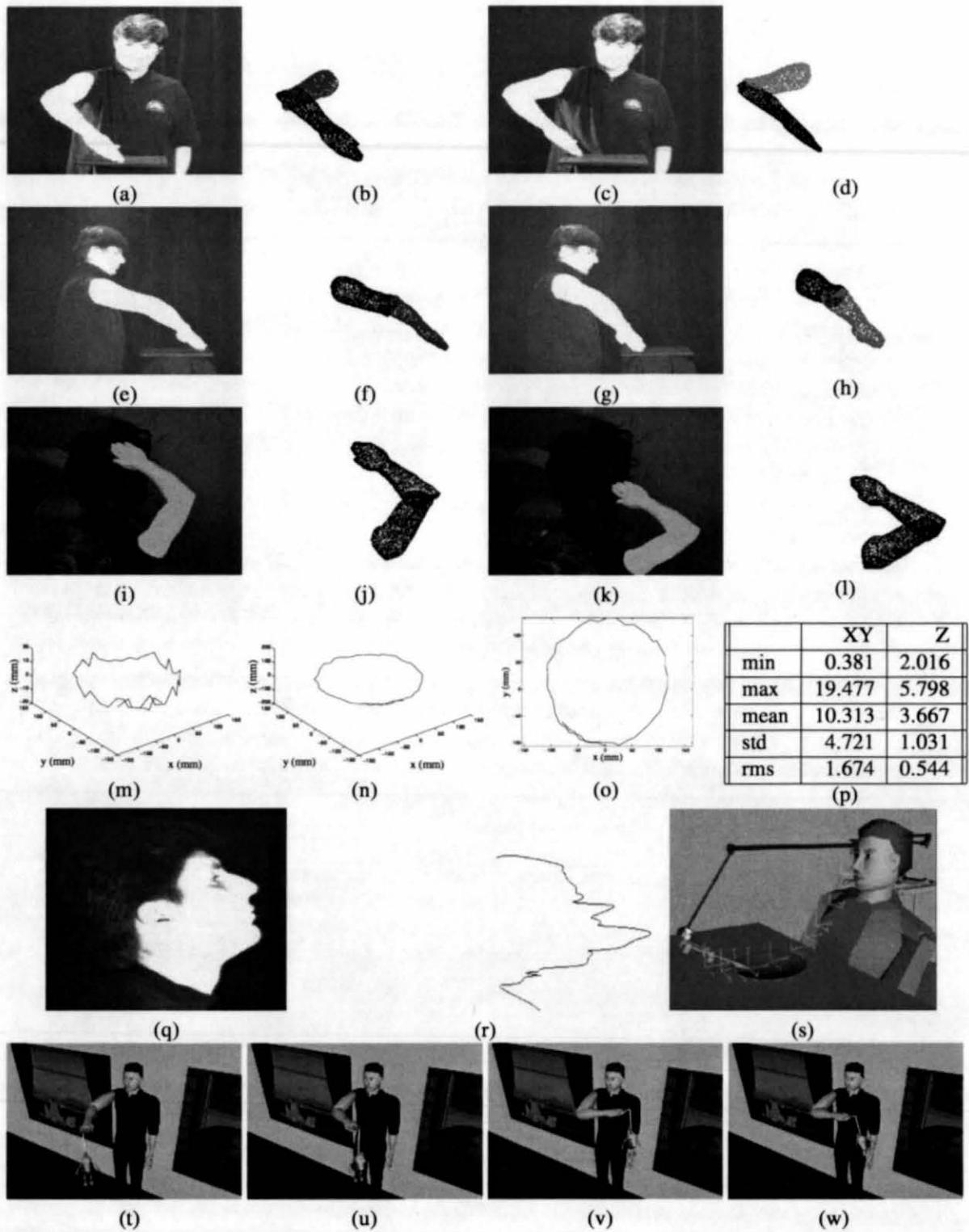


Figure 18: (a-l) Model-based three-dimensional tracking of the subject's arm. (m-p) The estimated trajectory of the tip of the right hand of the subject. (q-s) Virtual optimization of an experimental feeding mechanism. (t-w) The dynamic movement of the puppet.

Such applications include industry ergonomics crash simulations military training sports medicine and virtual actor design Industrial ergonomists and military trainers for example often need information regarding a workers posture gait and set of motions during the course of his/her job so that potentially hazardous postures and motions can be identified and prevented the productivity is increased and the most efficient way of training can be achieved

Various methods for controlling robotic and a virtual human s motion have been developed In the robotics community Raibert [58] generated a monopeded hopping robot motion that was able to keep its balance while hopping and jumping in the air Miura [46] studied human walking by developing a control algorithm for the dynamic walk of a biped Choi [18] developed a control scheme for a hexapod walking machine Adachi [1] implemented a dynamic control algorithm for a four legged walking robot In computer graphics to allow for real time and realistic looking human animations Bruderlin[14] generated a method for human walking using his KLAWE system a hybrid approach which combines goal directed and dynamic motion control The dynamic model used in this work was simplified so that both the stance and swing parts are modeled as an open chain system In the same paper the initial design of the open chain system was further simplified through the introduction of the virtual leg which was used to kinematically update the visual output Recently Tsutsuguchi [65] extended such a animation technique of flat plane walking to adapt to the changing terrain environment Hodgins [28] extended the earlier work of Raibert [58] on the use of PD control for robot hopping to the case of human walking running platform diving and vaulting The above methods for virtual human animation are based on the use of open loop dynamics and simple control strategies However there are other types of human activity where open loop dynamics is not sufficient For example ladder climbing an activity often used in industrial environments and in everyday life requires the use of closed-loop dynamics

In this section a method for animating ladder climbing is developed Like human walking ladder climbing can be divided into two basic phases

- 1 the stance phase where the involved limbs (legs and arms) support and propel the torso to move forward
- 2 and the swing phase where the involved limbs swing forward to grab the new supporting locations for the next stance phase based on potential energy

Unlike human walking or running during the stance phase in ladder climbing (and climbing in general) the torso one arm and one leg across the sagittal plane form a closed chain mechanism The swing phase like in human walking is modeled as an open loop chain mechanism Therefore the dynamic animation of ladder climbing requires the modeling of dynamic open and closed loop chains which is more challenging in terms of dynamics and control We will use the Lagrange multiplier method to formulate the dynamics of the stance phase which results in a system of differential algebraic equations (DAE) and the Lagrange method for the dynamic formulation of the swing phase

The input to the algorithm is a given forward velocity step length step frequency and a chosen gait The algorithm then determines kinematically the motion pattern i.e. the timing for the stance phase double support and swing phases For the purpose of this work, a periodic sinusoidal gait is assumed We therefore determine the initial and final position of the human during each phase of climbing These



Figure 19: a) Human model used in ladder climbing, b) Swing arm, c) Swing leg.

positions are fed into the dynamic formulation of our system in order to determine the motion of the human between the initial and final positions. We use an iterative numerical analysis technique which is based on the Newton-Raphson method to find a vector of joint torques that drives the dynamic system from the initial position to the final position. The stability problem during the iterative numerical integration is addressed by applying the Baumgarte stabilization [10] method. Finally, we demonstrate our real-time formulation through an animation experiment of ladder climbing.

We will now present the details of our method which consists of the geometric, dynamic and control formulations.

4.1 Geometric model

We geometrically define the human model and the ladder using *Jack*[®], the human modeling and simulation package developed at the University of Pennsylvania [56, 5]. An example of a simplified human model and virtual ladder (both generated in the Jack environment), used in the simulation are shown in Fig.19.

The human model is composed of nine articulated rigid body segments, i.e., the torso, the left and right arms and the legs. The relative coordinates are chosen to describe the position state of our model shown in Fig.20. Fig.20(a) shows the variables for the stance phase. Figs.20(b),(c) show the variables used during the swing phase for the arm and the leg, respectively. Therefore, the relative angular positions and their first derivatives are chosen to be our model's state variables.

4.2 Constraints in the Generation of the Climbing Motion

The program automatically calculates the geometric and kinematic constraints that will be used in the dynamic motion generation based on a given gait that matches the given step length and step height. During the process of climbing each step, we define constraints on the state variables for the initial and final human position, the duration of the stance, swing, and double support phases, and the avoidance of collisions between the human and the ladder. In particular, we have the following three steps:

1. Initial and final conditions for the state position variables

Fig. 20(a) indicates the beginning and the end of the current step during the stance phase. During the stance phase, the state position variables are not independent (see following sections) and therefore the dependent variables must satisfy the constraints formulated in (61) for the modeled mechanism. Finally, Figs. 20(b, c) indicate the beginning and the end configuration of the arms and the legs during the swing phase.

2. Duration of the current step for the stance, swing, and double support phases

These are calculated based on the same methodology used in [14] for the case of human walking. Assume that t denotes the duration of the simulation, t_{step} the duration of one step, and t_{ds} the duration of the double support state. Then, we have the following relationships:

$$t_{step} = t_{stance} - t_{ds} \quad (56)$$

$$t_{step} = t_{swing} - t_{ds} \quad (57)$$

$$t_{step} = \frac{d}{v} \quad (58)$$

$$t_{ds} = 0.25t_{step} \quad (59)$$

where d is the step length and v is the velocity of the motion.

3. Obstacle avoidance

To avoid possible collision of the human with the ladder's bars, the hand and the foot during the swing phase are constrained to be on polynomial curves whose coefficients are determined by the positions of the respective initial, final, and highest mid-points, as well as the slopes of the mid- and endpoints.

4.3 Dynamics

The Lagrangian formulation is used here to simulate the rigid parts of the human model. We make the following design decisions and approximations:

1. During the simulation, we consider the following phases: The stance and the swing phases, as shown in Fig. 20(a, c). Because the swing limbs' weight is small compared to the whole body, the separation does not disturb the total system. Therefore, the dynamics of the stance part and swing part can be formulated separately, with the roots of the swing limbs connected to the stance part.

2 For the purposes of our dynamic animation we make the simplified assumption that the motion is restricted in the sagittal plane

During the stance phase we model the human dynamically as a closed chain system comprised of the arm the torso and the leg (see Fig 20) The model's generalized coordinates $q = (q_1 q_2 q_3 q_4 q_5)^T$ (see Fig 20(a)) are not independent since from mechanism analysis [26] the mobility (number of independent degree of freedom) is three for this planar mechanism This means that the system configuration can be determined by fixing any three joint angle variables among the total of five

The two constraint equations for the closed chain have the form

$$\Phi(q, t) = 0 \quad (60)$$

where

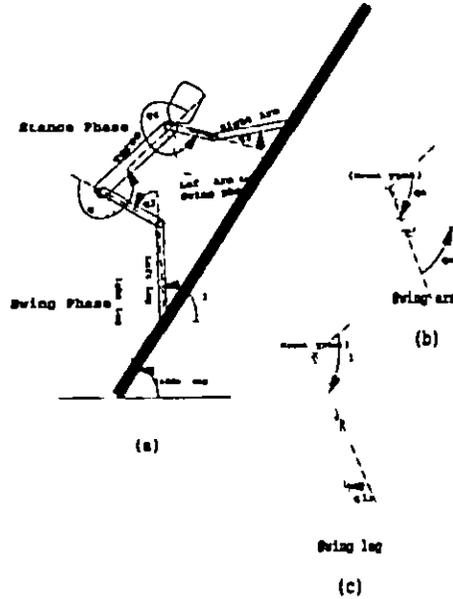


Figure 20 Static Postures

$$\Phi(q) = \begin{pmatrix} \sum_{i=1}^n l_i C_i - n_{step} * l_{step} * \cos(\alpha) \\ \sum_{i=1}^n l_i S_i - n_{step} * l_{step} * \sin(\alpha) \end{pmatrix} = 0 \quad (61)$$

$C_i = \cos(q_i^{abs})$, $S_i = \sin(q_i^{abs})$ and q_i^{abs} is the absolute angle of q_i with respect to the inertial frame

Given that the number of our model's state variables are greater than the number of degrees of freedom our system has actuator redundancy [48] and is under determined To compute the dynamics of such a system we use the Lagrange multiplier method which consists of the following formulation

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}} \right) - \frac{\partial L}{\partial q} + \Phi_q^T \lambda = Q_{ex} \quad (62)$$

where $\Phi_q^T \lambda$ represents the constraint force that replaces the removed joints so that the formulation for the open chain mechanism can be adopted in the closed loop chain system

Q_{ex} is the vector of external torques for each joint. From biomechanics experimental studies [32-14] the torque pattern within a step can be approximated by an exponentially damped force function in the form of Ae^{-at} . Joint elasticity and compliance can be modeled respectively as $B(q - q_0)$ and $C(q - q_0)$

(62) represents five second order differential equations and (5+2) unknowns (2 stands for the constraint equations). The five components of vector q and the two elements of vector λ . In order to have a sufficient number of equations it is necessary to supply two more equations. The obvious choice is to use the algebraic constraint equation (61) together with (62) to form a set of differential algebraic equations (DAEs). By differentiating (61) twice with respect to time we have

$$\Phi_{qq} \ddot{q} = -\ddot{\Phi}_t - \Phi_{q\dot{q}} \dot{q} = c \quad (63)$$

Thus

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{pmatrix} q \\ \lambda \end{pmatrix} = \begin{pmatrix} Q \\ c \end{pmatrix} \quad (64)$$

which is a system of (5+2) equations with (5+2) unknowns whose matrix is symmetric and positive definite and sparse in many practical cases. Such a matrix equation can be rewritten as a first order state system. By inverting such a matrix system we can numerically integrate this dynamic system. Due to possible numerical instability $\Phi = 0$ may be violated during the numerical integration. We therefore use the Baumgarte [10] stabilization method. Basically this method replaces (61) with

$$\Phi + 2\alpha\dot{\Phi} + \beta^2\ddot{\Phi} = 0 \quad (65)$$

Thus the numerical solution will be asymptotically stable in the sense of Ljapunov. From our experience choosing $\alpha = \beta = 7$ gets the best results (fast stabilization).

By using (65) instead of (61) the matrix form differential algebraic equation becomes

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{pmatrix} q \\ \lambda \end{pmatrix} = \begin{pmatrix} Q \\ g \end{pmatrix} \quad (66)$$

where $g = -\ddot{\Phi}_t - \Phi_{q\dot{q}} \dot{q} - 2\alpha(\dot{\Phi}_{q\dot{q}} + \dot{\Phi}_t) - \beta^2\ddot{\Phi}$. By defining the state variable $x = [q^T \quad \dot{q}^T]^T$ we can solve the state vector by integrating the dynamic equation numerically using a standard Runge-Kutta method [57]. The evaluation of the matrices within the matrix equation can be done symbolically using Maple [40].

During the swing phase we use an unconstrained Lagrangian formulation and with generalized coordinates $q = [x_{root}, y_{root}, \theta_1, \theta_2]^T$. The dynamic system of equation has the form

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}} \right) - \frac{\partial L}{\partial q} = Q_{ex} \quad (67)$$

Notice that $x_{root}, y_{root}, \dot{x}_{root}, \dot{y}_{root}, \ddot{x}_{root}, \ddot{y}_{root}$ are obtained from the stance phase

4.4 Control

If the forces acting on an object are completely specified through the course of the simulation, the object's position and velocity can be computed by integrating the equations of motion over time. This is called forward dynamics. Forward dynamics is sufficient for the simulation of simple or passive systems, but it can not be used for our problem because of its lack of control. It is virtually impossible for the user to guess the forces required to drive the articulated object to the desired initial and final states.

Here the forces and torques that drive the dynamic model of the system are determined by a numerical technique. Typically, the problem of finding the generalized forces which drive the dynamic system from the initial state to the final state of the current step can be written as

$$M\ddot{\mathbf{q}} = \mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}}, \tau), \quad (68)$$

subject to $\mathbf{q}(t_0) = \mathbf{q}_0$ and $\mathbf{q}(t_f) = \mathbf{q}_f$, for $t_0 \leq t \leq t_f$.

The solution of the above dynamic system can be accomplished numerically by integrating the equations of motion iteratively over the duration t_{step} , and by modifying \mathbf{Q}_{ex} on each iteration, until the state variable satisfies the final condition. The Newton-Raphson method [57] is used here to solve such a system.

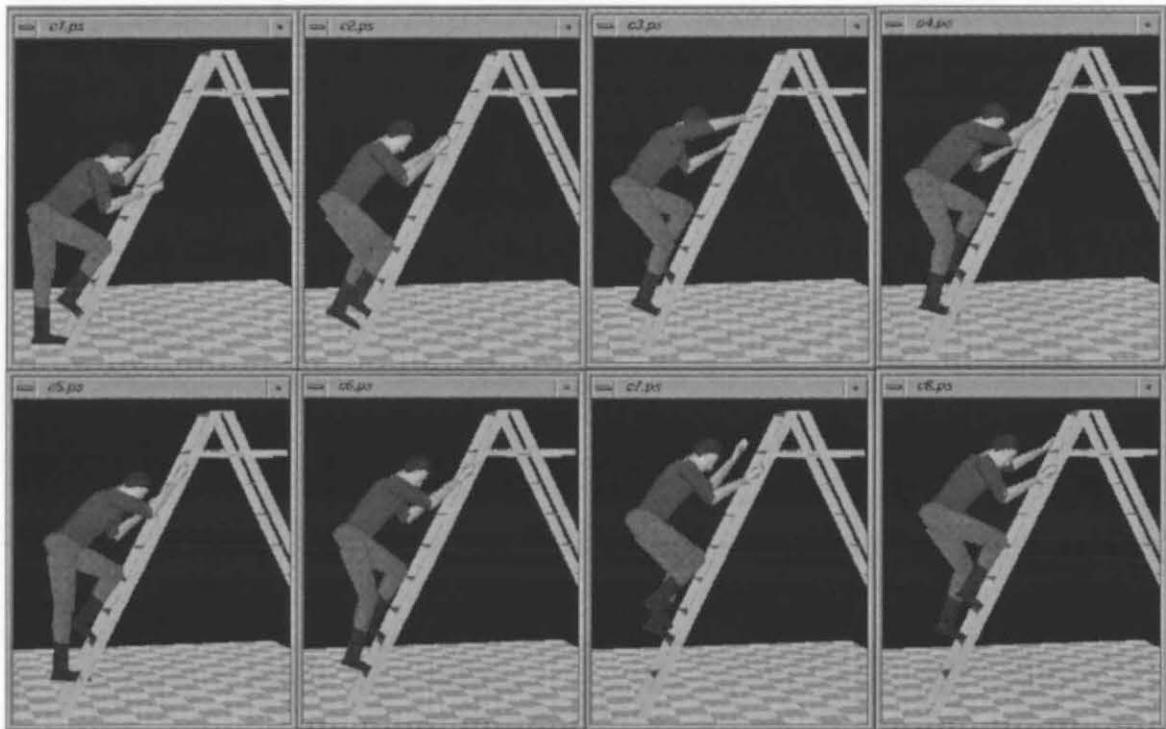


Figure 21: Animation sequence of a human climbing a simple ladder.

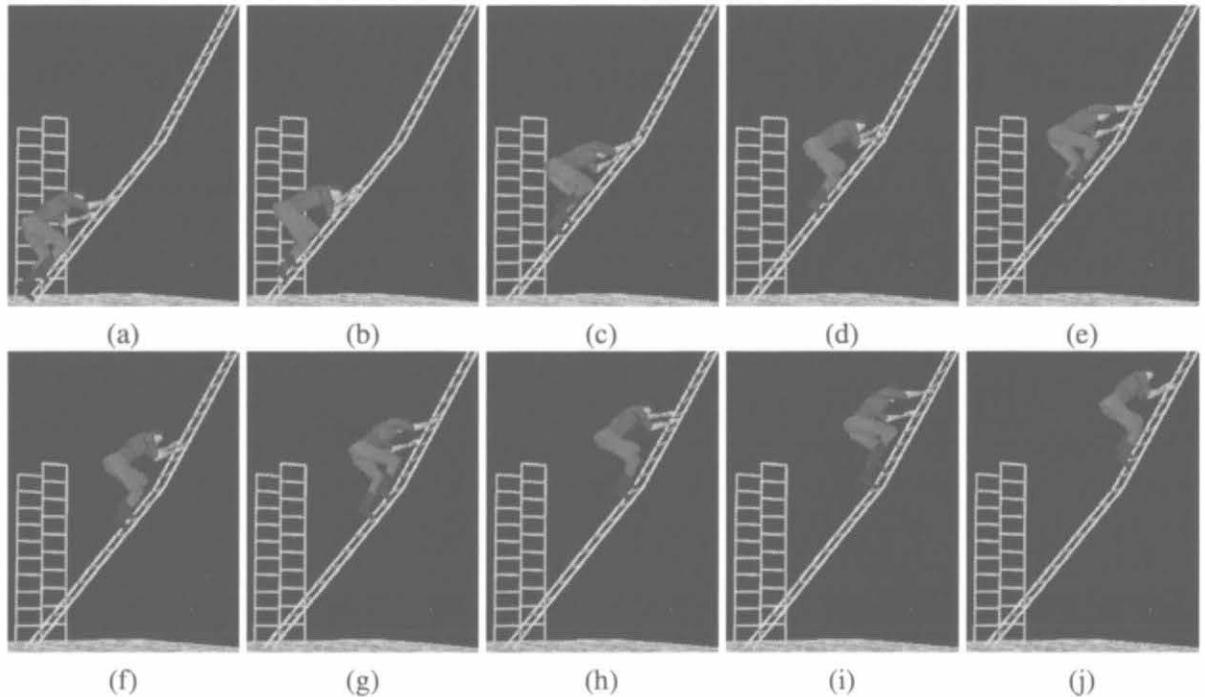


Figure 22: (a-j) Animation sequence from a human climbing a ladder with variable inclination and variable step length. Notice the change in the inclination of the torso as the human climbs the steeper portion of the ladder.

4.5 Results

The graphics animation results we have obtained for ladder climbing have been simulated using the Jack[®] system. We have conducted several experiments involving ladders with differing steps. The experiments run at approximately 5Hz on an SGI Indigo 2 Impact with an R4400 processor.

In Fig.21 we show an automatically generated climbing motion sequence of a simple ladder. Based on the kinematic user-defined input mentioned in the previous sections, we initialize our dynamic human figure on the ladder. Then the algorithm automatically calculates, based on the previously presented dynamic control algorithm, the motion of the human climbing the ladder steps. Even though, it is not needed in this application, an additional advantage of the method is the calculation of the joint torques. Such torques are useful in other applications such as military training.

In Fig.22(a-j) we show an animation from a human climbing a ladder with variable inclination and step length. For the lower portion of the ladder the step length was 28cm, while for the upper portion it was 26cm. In the figures (and in the video sequence) we demonstrate the two step lengths with the two ladders in the background. Based on the kinematic user-defined input mentioned in the previous sections, we initialize our dynamic human figure on the ladder. Then the algorithm automatically calculates, based on the previously presented dynamic control algorithm, the motion of the human climbing the ladder steps. For the lower portion of the ladder where the inclination and the step length

are the same the dynamic solution for each step of the lower part of the ladder is the same. However when the human attempts to climb the upper part of the ladder where the inclination and the step length change the dynamic solution adapts to the changes in the initial and the final postures of the human. As a result notice the change in the inclination of the human's torso and limbs.

We have also tested our method with many other ladders of variable step length and inclination. In all the cases our solution produced successful results.

5 Discussion

We have presented a variety of physics based approaches towards the dynamic control of animations involving articulated objects with open and closed loop kinematic chains. We have applied these approaches to produce a variety of human animations. To achieve this we have employed a variety of methods from kinematics, dynamics and control theory. Even though dynamic simulations are becoming increasingly popular in computer animation to generate more realistically looking motion, the development of *real time* dynamic and control methods for animations and behavior modeling is still a challenge. We expect that in the years to come these methods will become increasingly popular and very realistic results will be obtained.

References

- [1] Adachi H, Koyachi N and Nakano E. Mechanism and control of a quadruped walking robot. *IEEE Control Systems Magazine* (Oct 1988) pp 14-19.
- [2] K Akita. Image sequence analysis of real world human motion. *Pattern Recognition* 17:73-83 1984.
- [3] W W Armstrong and M Green. The dynamics of articulated rigid bodies for purposes of animation. In M Wein and E M Kidd editors. *Graphics Interface 85 Proceedings* pages 407-415. Canadian Inf Process Soc. 1985.
- [4] A Azarbayejani, T Starner, B Horowitz and A Pentland. Visually controlled graphics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(6):602-605 June 1993.
- [5] N I Badler, C B Phillips and B L Webber. *Simulating humans: computer graphics animation and control*. Oxford University Press, New York, NY 1993.
- [6] D S Bae and E J Haug. A recursive formulation for constrained mechanical system dynamics. Part I: open loop systems. *Mech Struct & Mach* 15(3):359-382 1987.
- [7] D Baraff. Fast contact force computation for nonpenetrating rigid bodies. In Andrew Glassner editor. *Proceedings of SIGGRAPH 94 (Orlando Florida July 24-29 1994)*. Computer Graphics Proceedings Annual Conference Series pages 23-34. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0 89791 667 0.

- [8] D Baraff Linear-time dynamics using lagrange multipliers In *Proceedings of SIGGRAPH 96 Computer Graphics Proceedings Annual Conference Series* pages 137–146 ACM SIGGRAPH ACM Press August 1996
- [9] R Barzel and A H Barr A modeling system based on dynamic constraints In John Dill editor *Computer Graphics (SIGGRAPH 88 Proceedings)* volume 22 pages 179–188 August 1988
- [10] J Baumgarte Stabilization of constraints and integrals of motion in dynamical systems *Computer Methods in Applied Mechanics and Engineering* 1972
- [11] A G Bharatkumar K E Daigle M G Pandy Q Cai and J K Aggarwal Lower limb kinematics of human walking with the medial axis transformation In *Proceedings of the 1994 IEEE Workshop on Motion of Non Rigid and Articulated Objects* pages 70–76 Austin TX November 11 12 1994 IEEE Computer Society Press
- [12] Y Y M J Black S X Ju A parameterized model of articulated image motion In *Proceedings of the Second International Workshop on Automatic Face and Gesture Recognition* pages 38–44 Killington Vermont October 14-16 1996
- [13] C Bregler and J Malik Video motion capture Technical Report UCB/CSD 97 973 UC Berkeley 1997
- [14] A Bruderlin and T Calvert Goal directed dynamic animation of human walking In *Computer Graphics (Proc SIGGRAPH)* volume 23 pp 233–242 ACM July 1989
- [15] A Bruderlin and L Williams Motion signal processing In *Proceedings of SIGGRAPH 95 Annual Conference Series* pages 97–104 Los Angeles CA August 6 11 1995 ACM SIGGRAPH Addison Wesley
- [16] T K Capin I S Pandzic H Noser D Thalmann and N M Thalmann Virtual human representation and communication in vlnet. *IEEE Computer Graphics and Applications* 17(2) 42–53 March April 1997
- [17] Z Chen and H J Lee Knowledge guided visual perception of 3D human gait from single image sequence *IEEE Transactions on Systems Man and Cybernetics* 22(2) 336–342 1992
- [18] Choi B S and Song S M Fully automated obstacle crossing gaits for walking machines *Proc of the 1988 IEEE Int Conf on Robotics and Automation*. v2 April 24 29 1988 pp 802 807
- [19] R Featherstone *Robot dynamics algorithms* Kluwer Academic Publishers 1987
- [20] W Frey M Zyda, R McGhee and W Cockayne Off the-shelf real time human body motion capture for synthetic environments Technical Report NPSCS 96 003 Computer Science Department, Naval Postgraduate School Monterey CA June 1996

- [21] D M Gavrilu and L S Davis 3 D model based tracking of humans in action a multi view approach In *Proceedings of the 1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* pages 73–80 San Francisco CA June 18 20 1996
- [22] A Gelb *Applied Optimal Estimation* MIT Press Cambridge MA 1974
- [23] M Gleicher and P Litwinowicz Constraint based motion adaptation Technical Report TR 96 175 Apple Computer CA June 1996
- [24] H Goldstein *Classical Mechanics* Addison Wesley 1950
- [25] L Goncalves E D Bernardom E Ursella and P Perona Monocular tracking of the human arm in 3D In *Proceedings of the Fifth International Conference on Computer Vision* pages 764–770 Boston MA June 20 22 1995
- [26] Greenwood D T Principles of Dynamics Prentice Hall 1988
- [27] Y Guo G Xu and S Tsuji Understanding human motion patterns In *Proceedings of the 12th International Conference on Pattern Recognition* pages B 325–329 Jerusalem Israel October 9 13 1994
- [28] J K Hodgins W L Wooten D C Brogan and J F O'Brien Animating human athletics In Robert Cook editor *SIGGRAPH 95 Conference Proceedings* Annual Conference Series pages 71–78 ACM SIGGRAPH Addison Wesley August 1995 held in Los Angeles California 06 11 August 1995
- [29] D Hogg Model based vision A program to see a walking person *Image and Vision Computing* 1(1) 5–20 1983
- [30] D P Huttenlocher J J Noh and W J Rucklidge Tracking non rigid objects in complex scenes In *Proceedings of the IEEE Forth International Conference on Computer Vision* Berlin Germany May 11 14 1993
- [31] J Cohen M Lin D Manocha and K Ponamgi I COLLIDE An interactive and exact collision detection system for large scaled environments In *Proceedings of ACM Int 3D Graphics Conference* pages 189–196 1995
- [32] Inman V T Ralston H J and Todd F Human Walking Williams & Williams Baltimore 1981
- [33] I A Kakadiaris *Motion Based Part Segmentation Shape and Motion Estimation of Multi Part Objects Application to Human Body Tracking* PhD dissertation Department of Computer Science University of Pennsylvania, Philadelphia PA October 1996
- [34] I A Kakadiaris and D Metaxas 3D Human body model acquisition from multiple views In *Proceedings of the Fifth International Conference on Computer Vision* pages 618–623 Boston MA June 20 22 1995

- [35] I A Kakadiaris and D Metaxas Model based estimation of 3D human motion with occlusion based on active multi viewpoint selection In *Proceedings of the 1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* pages 81–87 San Francisco CA June 18-20 1996
- [36] V Krovi V Kumar G K Ananthasuresh and J M Vezién Design and virtual prototyping of rehabilitation aids Accepted for Publication 1997 ASME Design Engineering Technical Conferences 14 17 September 1997 1997
- [37] H J Lee and Z Chen Determination of 3D human body postures from a single view *Computer Vision, Graphics and Image Processing* 30 148–168 May 1985
- [38] K Lilly *Efficient Dynamic Simulation of Robotic Mechanisms* Kluwer Academic Publishers 1993
- [39] P Maes B Blumberg T Darrell and A Pentland The alive system Full body interaction with autonomous agents In *Proceedings of Computer Animation 95 Conference* 1995
- [40] Maple a symbolic computation software developed by Waterloo Maple Software Canada
- [41] D Marr and K Nishihara Representation and recognition of the spatial organization of three dimensional shapes *Proceedings of the Royal Society of London* 200 269–294 1978
- [42] D Metaxas and D Terzopoulos Shape and nonrigid motion estimation through physics based synthesis *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(6) 580–591 June 1993
- [43] D Metaxas and D Terzopoulos Dynamic deformation of solid primitives with constraints In Edwin E Catmull editor *Computer Graphics (SIGGRAPH 92 Proceedings)* volume 26 pages 309–312 July 1992
- [44] D Metaxas *Physics based Deformable Models Applications to Computer Vision, Graphics and Medical Imaging* Kluwer Academic Publishers 1996
- [45] M McKenna and D Zeltzer Dynamic simulation of autonomous legged locomotion In *Computer Graphics (SIGGRAPH proceedings)* ACM August 1990
- [46] Miura, H and Shimoyama, I Dynamic Walk of a Biped *Internat J Robotics Res* v3 1984 pp 60 74
- [47] M Moore and J Wilhelms Collision detection and response for computer animation In John Dill editor *Computer Graphics (SIGGRAPH 88 Proceedings)* volume 22 pages 289–298 August 1988
- [48] Nakamura, Y *Advanced robotics* Addison Wesley Publishing Company New York 1991

- [49] S A Niyogi and E H Adelson Analyzing gait with spatiotemporal surfaces In *Proceedings of the 1994 IEEE Workshop on Motion of Non Rigid and Articulated Objects* pages 64–68 Austin TX November 11–12 1994 IEEE Computer Society Press
- [50] J O Rourke and N I Badler Model based image analysis of human motion using constraint propagation *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2(6) 522–536 1980
- [51] N Orlandea M A Chace and D A Calahan A sparsity oriented approach to dynamic analysis and design of mechanical systems robotic mechanisms *ASME Journal of engineering for industry* pages 773–784 1977
- [52] A Pentland Automatic extraction of deformable part models *International Journal on Computer Vision* 4 107–126 1990
- [53] A Pentland Machine understanding of human action In *7th International Forum on Frontier of Telecommunication Technology* Tokyo Japan November 1995
- [54] A Pentland and S Sclaroff Closed form solutions for physically based modeling and recognition *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13(7) 715–729 1991
- [55] F Pfeiffer and C Glocker *Multibody Dynamics with Unilateral Contacts* Wiley 1996
- [56] C B Phillips and N I Badler Interactive behaviors for bipedal articulated figures In Thomas W Sederberg editor *Computer Graphics (Proc SIGGRAPH 91)* volume 25 pp 359–362 July 1991
- [57] W Press B Flannery S Teukolsky and W Vetterling *Numerical Recipes The Art of Scientific Computing* Cambridge University Press 1986
- [58] Raibert M H Hopping in legged systems modeling and simulation for the 2D ones legged case *IEEE Trans Systems Man Cybernet* 14(3) (1984) pp 451–463
- [59] M H Raibert and J K Hodgins Animation of dynamic legged locomotion In Thomas W Sederberg editor *Computer Graphics (SIGGRAPH 91 Proceedings)* volume 25 pages 349–358 July 1991
- [60] J M Rehg and T Kanade Visual tracking of high DOF articulated structures An application to human hand tracking In J O Eklundh editor *Proceedings of the Third European Conference on Computer Vision* pages 35–46 Stockholm Sweden May 2–6 1994 Springer Verlag
- [61] J M Rehg and T Kanade Model based tracking of self occluding articulated objects In *Proceedings of the Fifth International Conference on Computer Vision* pages 612–617 Boston MA June 20–22 1995
- [62] K Rohr Towards model based recognition of human movements in image sequences *Computer Vision Graphics and Image Processing Image Understanding* 59(1) 94–115 Jan 1994

- [63] A Shabana *Computational Dynamics* Wiley 1994
- [64] A State G Hirota, D T Chen W F Garrett and M A Livingston Superior augmented reality registration by integrating landmark tracking and magnetic tracking In *Proceedings of SIGGRAPH 96 Annual Conference Series* pages 429–438 New Orleans LA August 4-9 1996 ACM SIGGRAPH Addison Wesley
- [65] Tsutsuguchi K Sakano H and Watanabe Y Terrain Adaptive Human Walking Animation *Systems and Computers in Japan* v26 n5 1995 pp 79 88
- [66] J M Vézien V Kumar R Bajcsy R Mahoney and W Harwin Design of customized rehabilitation aids In *Medical Robots* October 1996
- [67] S Wachter and H H Nagel Tracking of persons in monocular image sequences In *Proceedings of IEEE Nonrigid and Articulated Motion Workshop* pages 2–9 June 16 1997
- [68] M W Walker and D E Orin Efficient dynamic computer simulation of robotic mechanisms *Trans ASME Jnl Dynamic Systems Measurement and Control* pages 205–211 1982
- [69] M Gleicher A Witkin and W Welch Interactive dynamics In *Proceedings of the Symposium on Interactive 3D Graphics* volume 24 pages 11–21 March 1990
- [70] A Witkin and Z Popović Motion warping In *Proceedings of SIGGRAPH 95 Annual Conference Series* pages 105–108 Los Angeles CA August 6 11 1995 ACM SIGGRAPH Addison Wesley
- [71] C Wren A Azarbayejani T Darrell and A Pentland Pfinder Real-time tracking of the human body In *Proceedings of the Second International Workshop on Automatic Face and Gesture Recognition* pages 51–56 Killington Vermont October 14 16 1996
- [72] J Wittenberg *Dynamics of Systems of Rigid Bodies* Teubner 1977
- [73] M Yamamoto and K Koshikawa Human motion analysis based on a robot arm model In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* pages 664–665 Lahaina, Maui Hawaii June 3–6 1991