UNIVERSITY OF PENNSYLVANIA

Computer Graphics Research Laboratory
Final Report

Grant #: DAAL03-90-G0198

December 1991

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	12/91	Final Report 1 Sep 90-31 Aug 91

4. TITLE AND SUBTITLE

Human Body Modeling & Simulation

5. FUNDING NUMBERS

DAAL03-90-G-0198

6. AUTHOR(S)

Dr. Norman I. Badler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Pennsylvania
Computer and Information Science Department
200 S. 33rd St.
Phila., PA 19104

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARO 28131.1-MA

11. SUPPLEMENTARY NOTES

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

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This final report includes descriptions of research projects underway in the Computer Graphics Research Lab at the University of Pennsylvania from 9/1/90 through 8/31/91 involving human body modeling and simulation.

14. SUBJECT TERMS

Human Body Modeling and Simulation

15. NUMBER OF PAGES

22

16. PRICE CODE

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

Contents

1 Introduction: Norman I. Badler 1

2 The Enhancement of the Fully Articulated Hands: Xinmin Zhao 3

2.1 Design and Implementation of a New Grasp System 3

2.2 Grasping as a Behavior 4

2.3 Improving the Hand Model 4

2.4 Error Recovery (Crash Free *Jack*) 4

2.5 Memory Usage Problem caused by the Collision Detection Routine: Xinmin Zhao . . 5

2.6 Known Problems 5

2.7 New Commands and Their Explanations 6

3 SASS: Francisco Azoula 8

4 Locomotion: Hyeongseok Ko 9

5 Statics Computation: Hyeongseok Ko 9

6 Spline Paths: Paul J. Diefenbach 10

7 Geometry Translators: Michael J. Hollick 10

7.1 *Pro/Engineer* "Render" format 10

7.2 *AutoCAD* "DXF" format 10

8 Multi Parameters Function Input for *Jack*: Bond-Jay Ting 11

9 Radiosity Rendering: Min-Zhi Shao 11

10 Texture and Ray-Tracing: Jeffry S. Nimeroff 12

11 Moving Posture Reconstruction: Jianmin Zhao 12

12 Virtual Reality: Rebecca Mercuri 14

13 Audio *Jack*: Kok-Hoon Teo 15

14 Optimal Mesh Algorithms for the Voronoi Diagram of Line Segments, Visibility Graphs and Motion Planning in the Plane: Suneeta Ramaswami 16

15 Flock of Birds Interface To *Jack*: Mike Hollick 17

15.1 Hardware/Operating System Information 17

15.2 Using the FOB in *Jack* 18

15.2.1 Commands 18

15.2.2 Interface Details 19

15.2.3 Tips for Use 19

15.3 Future Improvements 20

Computer Graphics Research Laboratory

Norman I. Badler
Department of Computer and Information Science
University of Pennsylvania
Philadlephia, PA 19104-6389

1 Introduction: Norman I. Badler

This Quarterly Report includes descriptions of various projects underway in the Computer Graphics Research Lab during April through June 1992. These reports include:

- The articulated hand model was improved considerably by making the grasping motions into a *Jack* behavior. This also includes new handshapes, and a compliant grip mode where (though slow) the hand may interactively adjust to the gripped object's shape as either moves.
- The Spreadsheet Anthropometric Scaling System is nearing release, and is being updated for the new body structure.
- The locomotion algorithm was greatly improved by basing the fundamental walk cycle on biomechanics data, while curved path capabilities remain intact.
- The static torque loads on any joint can now be reliably and interactively computed.
- Splined motion paths were added into the animation interface.
- There are new geometry translators for Pro/Engineer "Render" format and AutoCAD "DXF" format.
- Multi-parameter function inputs are being added so that *Jack* may eventually modify joint and segment shapes depending on posture or other inputs.

- A new, fast, progressive refinement, radiosity rendering algorithm is outlined.
- Texture rendering and ray-tracing are discussed.
- Progress in moving posture reconstruction is realized through a buffered analysis system where tracking errors may be detected and the analysis "backed-up" to reconstruct a proper 3-D sequence.
- Virtual reality research includes a "SoundWorld" environment and other acoustic and visual spaces.
- Audio object attributes were added so that objects or behaviors can have associated sounds.
- A short paper summarizing work on parallel algorithms for planar motion path planning is included.
- The Ascension Technologies Flock of Birds may be interfaced to *Jack*; the result is a "virtual" interactive human figure positioned by the multiple 6 degree of freedom "bird" sensors.

Much of this Quarter was spent preparing *Jack* for the Showcase real-time networking demonstration at SIGGRAPH '92. A description of networked *Jack* will be included in the next Quarterly Report

We have begun offering *Jack* Training seminars as part of the ongoing effort to make *Jack* both usable and viable for the Human Factors community. The first course will be held in September '92, the second in December '92 and the third in March '93. More information of these courses will be forthcoming in the next Quarterly Report.

For information on *Jack* licenses or training sessions please contact:

David Harding
Center for Technology Transfer, Suite 300
3700 Market St.
University of Pennsylvania
Philadelphia, PA 19104-3147
215-898-9585
215-898-9519 (Fax)

This research is partially supported by ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Human Engineering Laboratory, Natick Laboratory, and NASA Ames Research Center; U.S. Air Force DEPTH contract through Hughes Missile Systems F33615-91-C-0001; MOCO Inc.; and NSF CISE Grant CDA88-22719. Partial support for this work, especially Undergraduate contributions, was provided by the National Science Foundation's Instrumentation and Laboratory Improvement Program through Grant #USE-9152503.

2 The Enhancement of the Fully Articulated Hands: Xinmin Zhao

2.1 Design and Implementation of a New Grasp System

When I tried to use the existing grasp system to perform several conventional grasping tasks, the results were not satisfactory. One of the main problems is that it usually cannot place the hand/fingers a the 'right' position/orientation to perform the grasp successfully.

To solve this problem, a new grasp system has been designed and implemented. The new grasp system provides various mechanisms to enable the user to position the hand/fingers at the desired position/orientation. In the new grasp system, it is also possible for the user to position each finger at a particular place using constraints to perform unusual grasp tasks. Yet another new feature is the concept of grasp by example. It is explained in the following:

> The user may teach the system how to do a particular kind of grasp by giving an example. The system will automatically create a new grasp type based on the example. For instance, the user can show the system how to grasp a cabinet door knob and new grasp type "door knob grasping" will be created automatically to do this kind of grasp.

Some of new features available in the new grasp system are briefly explained below:

a. *aligning palm direction*

To grasp objects successfully, the direction of the palm has to be aligned in a certain direction. For example, to grasp hammer handles using power grasp, the palm direction should be aligned with the handle's 'long' direction.

b. *aligning finger direction*

In order to prevent fingers from penetrating into the objects during grasping, the user may specify that fingers point to a desired direction.

c. *collision avoidance during reaching*

In the new grasp system, the hand will approach the target object from a given direction so that it will not go through it before grasping.

d. *fine tuning hand position*

The user may give displacement transforms to fine tune the grasping positions. This is especially useful if the desired hand/finger positions are not predefined points (e.g., middle of an edge).

The new grasp system has been used to perform a number of grasping tasks successfully (grasping hammers, door knobs, rings, urns, glasses, boxes, etc.). As a matter of fact, the new grasp will

always work if the constraints are solved successfully and most of time they are. Various techniques have also been employed to use the constraint solver both effectively and efficiently.

2.2 Grasping as a Behavior

The original grasp system was a stand-alone procedure. *Jack* has to wait until grasping is done, which may take a long time if collision detection is in action.

The new grasp system has been successfully integrated into *Jack* behavior system. Now the grasping is processed in the simulation procedure as all other behaviors are. Users of the new grasp system can do interactive grasping, in which case, if the target object is moved, the hand will automatically follow and catch it.

During interactive grasp, my original design was such that if the target were moved after the hand had been (partially) closed, the hand would be opened (to avoid collision with the object during catching) and moved to catch the target. A more useful case is where, the hand is not opened and is moved with the object without any delay. It is just like catching (grasping) a fish. If the fish moves after being caught, you may not want to open your hand to try to catch it again, even when there is a better grasp position. (Note that attaching human to the target using "attach figure" command will not work here because it will move the whole body with the moving target, rather than just the hand as the case in most interactive grasping).

Both types of interactive grasping have been implemented and the user has the option to use either one of them.

2.3 Improving the Hand Model

During grasping, the coordination of finger joint movement is critical to a successful grasp. For example, if finger tip joints move too fast, the grasp usually will not be successful. In the new grasp system, the coordination of finger joints movement has been implemented.

Another issue arises during grasping is the inter-finger coordination. For example, the coordination between thumb and index finger is very important to the success of the grasping operation. Some basic inter-finger coordinations have been implemented in the new grasp system.

2.4 Error Recovery (Crash Free *Jack*)

Like other large systems, *Jack* sometimes crashes. It is especially frustrating if the user didn't save his work before crash. Unfortunately, there is no way to predict when the system will crash, nor is there any simple way prevent crashes from happening. However, crash recovery can be done.

An Error Recovery System has been implemented which, after each crash, will be able to automatically restore the system to the state just before crashing. After error recovery, the user

can save his file and proceed 'as if' the crash never happened. The Error Recovery System is able to recover errors caused by all *Jack* commands.

What it does is:

a. save the system 'state' using Unix setjmp system call before executing each *Jack* command;

b. if it crashes during the execution of a command, restore the system to the pre-crash state using Unix longjmp system call;

The error recovery system has been intergrated into the *Jack* system and works very well as expected.

2.5 Memory Usage Problem caused by the Collision Detection Routine: Xinmin Zhao

The current collision detection routine available consumes too much memory (actually the memory it uses is unbounded and linearly increases with the number of collision detections is performed). Usually after a number of grasping tasks using collision detections, *Jack* will be killed because system runs out of memory.

This problem has been fixed (fortunately, a simple fix was adequate). Now the collision detection routine uses a fixed amount of memory (about 3 Kbytes), independent of the number of collision detections performed, compared to 10 Mbytes or more used before.

2.6 Known Problems

During reaching, sometimes the balance control cannot keep the body hold steady. As a result, the hands move with the body and sometimes they penetrate the target. However, this will not happen as long as the the body is held steady during reaching, and in most cases this is true. At this moment, it seems to me that the best thing to do to solve this problem is to improve the balance control of *Jack* .

During hand closing with collision detection option, only the collisions with the target segment are checked. In practice, this has proven to be inadequate if the target object is composed of multiple segments and the hand may collide with more than one of them. One way to solve this problem is to let user input the segments to be checked. Another way is to automatically check the adjacent segments.

The latter approach has been implemented: the program now automatically checks collision with the grasping segment and its adjacent segments in the same figure. Two segments are adjacent if and only if there is a joint connecting them.

2.7 New Commands and Their Explanations

turn error catcher on/off
new grasp
release object
create new grasp type by example
move finger
new close hand
turn grasp behavior on/off
turn interactive grasp behavior on/off
turn open hand on/off
turn approaching direction on/off
turn finger direction on/off
turn displacement on/off
turn collision detection on
turn collision detection off
change collision detection threshold
change touch site to tip
change touch site to base
change grasp constraint solving parameter
get joint angles
get hand joint angles
set move all dofs one step

The following are the commands that have been added to *Jack*:

1. *Turn Error Catcher on/off*

   This command turns on/off the error recovery mechanism described previously.

2. *New Grasp*

   This is the main procedure for grasp. In the new grasp system, there are 3 types of grasp. For the time being there are called *power grasp*, *precision grasp* and *cup grasp*. While power grasp behaves similar to the previous one, precision and cup grasp are new and their meanings are explained as follows:

Precision Grasp: In this type of grasp, user has control over the number of fingers to be used in the grasp, as well as the positioning of each individual finger on the grasped object. It is the most flexible type of grasp and should be able to perform any kind of grasp task desired. This type of grasp is entirely constraint based and techniques have been used minimize the interference of finger constraints (there is a constraint for each finger).

Cup Grasp: This type of grasp are useful to grasp a glass, cabinet knob, etc. The final shape of the hand is that all fingers come together if there is no object to be grasped. Collision detection is used to stop a finger when it touches the surface of the object. (Note: The current human figure has a difficult time to execute the cup grasp. The problem is that it is impossible to bring all fingers together to form cup grasp gesture. What I did was to change the human figure definition. More specifically, I changed the upper limit of the little finger's first dof of the first joint, from 22.5deg to 30deg. If you use another human figure, cup grasp may not work well if you do not change these parameter or the hand geometry is not updated.)

While cup grasp is collision detection driven, precision grasp does not use collision detection at all. Power grasp is in the middle. The user can turn on/off collision detection for power grasp tasks.

3. *Create New Grasp Type by Example*

   It is difficult to provide all grasp types needed by many different users. As a result, new mechanisms are provided to give user the ability to perform 'non-standard' grasp tasks. While the precision grasp (positioning each finger individually) is one way to achieve this, yet another way is to let user create his/her own grasp types. This may avoid the tedious job of positioning all the fingers every time a grasp is performed.

   In each type of grasp, what matters are the initial and end shape of the hand, or the displacements of joint dofs (degrees of freedom), as well as the the coordinations of inter-joint, inter-finger movements. In each grasp type, the related information is recorded in a grasp-type structure and the grasp functions access this information to perform a particular type of grasping.

   To create a new grasp type is very simple: the system asks the user to show how the grasp is to be performed and records all the information discussed above in a grasptype structure just like the built-in "power" grasp.

4. *Move Finger and New Close Hand*

   These commands allow user to control the movement of one or a group of fingers. New Close Hand moves all fingers instead of some of the fingers as Move finger does.

5. *Grasp Behavior Control Commands*

   For each grasp task to be performed, user has the options to specify:

   an approaching direction to avoid hand collision with the object;

   a finger direction to avoid finger penetration through the object;

a displacement to fine tune the final hand/finger position;

User may also specify if the grasping will be interactive (moving the hand with target), and if it is interactive, should the hand be opened to catch the target. The default is non-interactive.

For each grasp task, user has the option to use collision detection. Cup grasp type is an exception, in which full collision detection will be always used.

6. *change collision detection threshold*

   During hand closing, each joint is moving a certain amount (e.g., 10 degree) at a time. Let us call this parameter incr. If collision occurs, it does not simply stop moving as was done before. Instead, incr is reduced (to half). A joint will stop moving only if its incr is less than the threshold and collision occurs. This command allows users to adjust the threshold dynamically for different tasks. For example, during experiment stage, the user may wish to use large threshold for performance and later use smaller threshold for better results.

7. *change touch site*

   In precision grasp, each finger is individually placed on the object. The user has the option to place the tip or base site of the last finger joint in a specific position. This is useful in some grasp tasks where a specific part of finger is required to touch the target.

8. *change grasp constraint solving parameter*

   Grasp tasks can be divided into two stages: reaching the target and closing hand. Before closing the hand, we have to make certain that the hand is close enough. This command sets the parameters to be used to measure the "closeness" of the hand to the target. There are 3 parameters:

   Distance: The distance field stored in the constraint is used to measure the closeness of the hand to the target.

   Converging Factor: The difference between current and previous distance is used to measure the potential for further improvement.

   Max Iterations: The number of times the grasp constraint is to be solved before given up.

3 SASS: Francisco Azoula

SASS is almost ready for the first release. I developed the new **SASS** figure output, that is the Peabody file, scaled to given population values. Since there have been changes in the figure definition used by *Jack*, I am in the process of modifying **SASS** output to have a more flexible way of changing the figure definition without the need of going into **SASS** internal code. The problem is that the Peabody definition keeps constantly changing, and therefore, **SASS** output becomes outdated. So the idea is to build up a figure output that is easy to upgrade.

I tested **SASS** database and it is not working. The problem is in the C - Prolog interface. I think for future releases we must modify the database, (write it in C) rather than use a Prolog shell, because it would be faster, but more important, it would allow **SASS** to be completely autonomous.

Also, I wrote the first draft of the User's Manual for **SASS**. It contains the basic details of the program, including new features.

4 Locomotion: Hyeongseok Ko

Locomotion data from rotoscoping is of limited to a small number of subjects and strides, but our graphics figures are not limited in computer animation applications. Adapting from limited data to the general situation is non-linear and non-trivial. Moreover, the segmentwise comparison between a subject and a figure can be arbitrary, which makes the problem even more complex.

As a solution, we tried to extract general properties from the subject's movement and make the figure imitate the subject. This imitation should be restricted within a certain neighborhood. For example, we can expect a good imitation of $1.2h$ tall figure with $1.2sl$ step length out of h tall subject with sl step length. But if the same figure tries to step $0.1sl$, we can imagine the conflict between the imitation and the step length achievement. The range of the step length and/or the variety of walking patterns can be extended *incrementally* by collecting more data into the database.

We applied the curved path walking algorithm (developed in last quarters) to the straight line walking. The motion was, as expected, more natural and robust. Local stepping is required frequently as a transition from a motion to another motion. It is performed in a more random fashion than walking. The amount of turn can be very large, and the direction of movement can be backwards or sideways (crab walking), which shows a totally different kind of movement from normal walking. It was implemented by modifying the curved path walking algorithm.

The curved path walking together with the local stepping can provide a high level control of the locomotion. If a goal position and orientation of a figure are given, our system analyzes internally the displacement between the current and the final configurations, computes the intermediate step sequence, determines whether each step should be a walking step or a local step, and finally produces the walking animation sequence. The step sequence is obtained by an algorithm based on geometric considerations.

5 Statics Computation: Hyeongseok Ko

We built a robotics model out of the *Jack* human figure. Each joint of multiple degrees of freedom was decomposed into the primitive joints each with a single degree of freedom. Standard robotics technique were applied on this model to compute the kinematics and static torques of the body.

Loads can be attached or general external forces can be exerted on the body to experiment the internal torque change due to the attachments. The torques are visulized by a bar graph in a window.

6 Spline Paths: Paul J. Diefenbach

Work was completed on the spline path routines. A generic interpolation scheme was developed for creating figure or segment paths based on a series of interpolants. Hooks were included for expansion of the interpolation functions past the current linear and Catmull-Rom methods. This could be expanded to behavioral functions based on a series of associated parameters.

The path routines provide mechanisms for creating, moving, deleting, or changing the time of individual points along the path. Furthermore, due to the Object-Oriented approach of the coding, path data can be in any format with an associated interpolation function. More precisely, two functions can exist; one for translational components and one for rotational components. Rotations can be stored and operated on in a variety of formats, including Euler angles or quaternions. Display functions are also associated for display of the paths and data.

7 Geometry Translators: Michael J. Hollick

7.1 *Pro/Engineer* "Render" format

The utility `pro2fig` converts a *Pro/Engineer* **Render** format file to a Peabody figure. Usage is: `pro2fig <render file>`. Each psurf file generated by the translator is named by appending `.pss` to the part name; the figure file name is based on the render file name, with its extension replaced with `.fig`.

Render files may be created in *Pro/Engineer* in either **Assembly** or **Part** mode. In Assembly mode, any number of part may be selected for inclusion into the render file. The density of surface facets may be controlled within *Pro/Engineer* by setting the "Quality Factor."

Information concerning the articulation of assemblies is not contained in render files. Thus, `pro2fig` creates a joint between each segment and a "base" segment that has no corresponding geometry. Vertex coordinates are assumed to be in centimeters, and color information is preserved.

7.2 *AutoCAD* "DXF" format

`dxf2pss` is used to convert a **DXF** file to a Peabody figure. (Note that a suitable naming convention for geometry translators has not yet been established!) Usage is: `dxf2pss <filename>.dxf`. Each layer will be placed in a separate psurf file. Each block will also be written to an individual psurf.

These psurf files will be named by appending `.pss` to the layer/block name. The resulting figure file will be `<filename>.fig`.

`dxf2pss` handles the majority of DXF entities, but it is not yet exhaustive. Supported entities include: blocks and insertions, 3Dfaces, 2D and 3D polylines, circles, arcs, and extrusions of each of these. Polygon meshes, dimensions, and text are currently not supported and will be ignored.

As in `pro2fig`, base joints must be created, since articulation information is not stored in the **DXF** format. Color information is very basic in **DXF**, so a random color is assigned to each segment.

8 Multi Parameters Function Input for *Jack*: Bond-Jay Ting

One way to solve deformation problem is to change nodes position dynamically. In my previous work, a single parameter polynomial function interface has been done. The work in this phase extends the interface from one input parameter to multiple input parameters. In most articulated motion, deformation occurs as a function of joint angle. The work in this stage uses degrees of freedom of joint angle as function parameters. Instead of using rigid body model, the deformation of contour body parts can be simulated by using function files.

9 Radiosity Rendering: Min-Zhi Shao

In this quarter, I basically developed a new radiosity rendering algorithm: the shooting and gathering progressive refinement method.

In the conventional radiosity method, the Gauss-Siedel method is used to obtain the solution of the simultaneous equations. This iterative approach converges to the solution by solving the system of equations one row at a time. The evaluation of the i'th row of the equations provides an estimate of the radiosity of patch i based on the current estimates of the radiosities of all other patch radiosities. In a sense, the light leaving patch i is determined by gathering the light from the rest of the environment.

It is possible, however, to reverse this process by determining the contribution made by patch i to the radiosity of all other patches. The reciprocity relationship provides the basis for reversing this relationship. This leads to the famous progressive refinement radiosity method, or the shooting method. Two major advantages over the traditional radiosity algorithm are:

1. an approximate image is produced in time linear to the number of patches
2. the $O(N^2)$ storage requirements for the form-factors have been eliminated.

The main idea of our method is to shoot as much light energy as possible from one set of form-factors calculation, i.e., one hemi-cube. Instead of shooting only the current light energy of the patch, we also gather the light energy from the rest of the environment to this patch and shoot it at the same time. By using the reciprocity relationship, one hemi-cube calculation can obtain both. In a sense, it means we get the second order approximation instead of first order approximation in the conventional progressive refinement method in each step of patch light energy shooting. Therefore, a faster convergence is expected. From the computing point of view, this also leads to a two pass approximation structure, light patch shooting and ordinary patch shooting.

Currently, we are implementing and documenting this algorithm. Details will appear in the next quarterly report.

10 Texture and Ray-Tracing: Jeffry S. Nimeroff

During the second quarter of 1992, I led a group of graduate students in the creation of a class-based $C++$ ray tracer designed to be the prototype that would ultimately take the place of the current *Jack* ray tracer. The renderer was designed hierarchically and its shading algorithm was based on the Brown Animation System lighting model. A simple texture mapping module was designed to allow easy installation of my spline based texture reconstruction and resampling algorithm.

My current work involves more research on texture reconstruction specifically for ray tracing algorithms. Since a ray tracer does not maintain the area of the projected pixel, the key to solving the problem appears to be the creation of a correct sampling distribution for texture space based on the transformation from eye space through object space into texture space. If this distribution can be constructed, stochastic sampling schemes can be used to anti-alias the ray traced texture mapped image.

I also plan to start my dissertation research on real-time rendering of complex scenes. Since known algorithms are compute intensive and cannot process and render complete datasets in real time, the solution to this problem appears to be one of correctly sampling the input data (decimation) and constructing incremental algorithms to perform the rendering.

11 Moving Posture Reconstruction: Jianmin Zhao

I have almost finished the first version of the program *Moving Posture Reconstruction.* Conceptually, the program consists of several modules as shown in Figure 1.

A parser parses image data in plain text form and sends parsed image frames to a buffer. The *Control* then takes the pointer to the buffer and passes it to the *Critical condition monitor* to detect whether a critical frame has been encountered. The monitor is a finite state automata, which stores the state of each frame in the same buffer. The monitor can only detect a critical frame after a

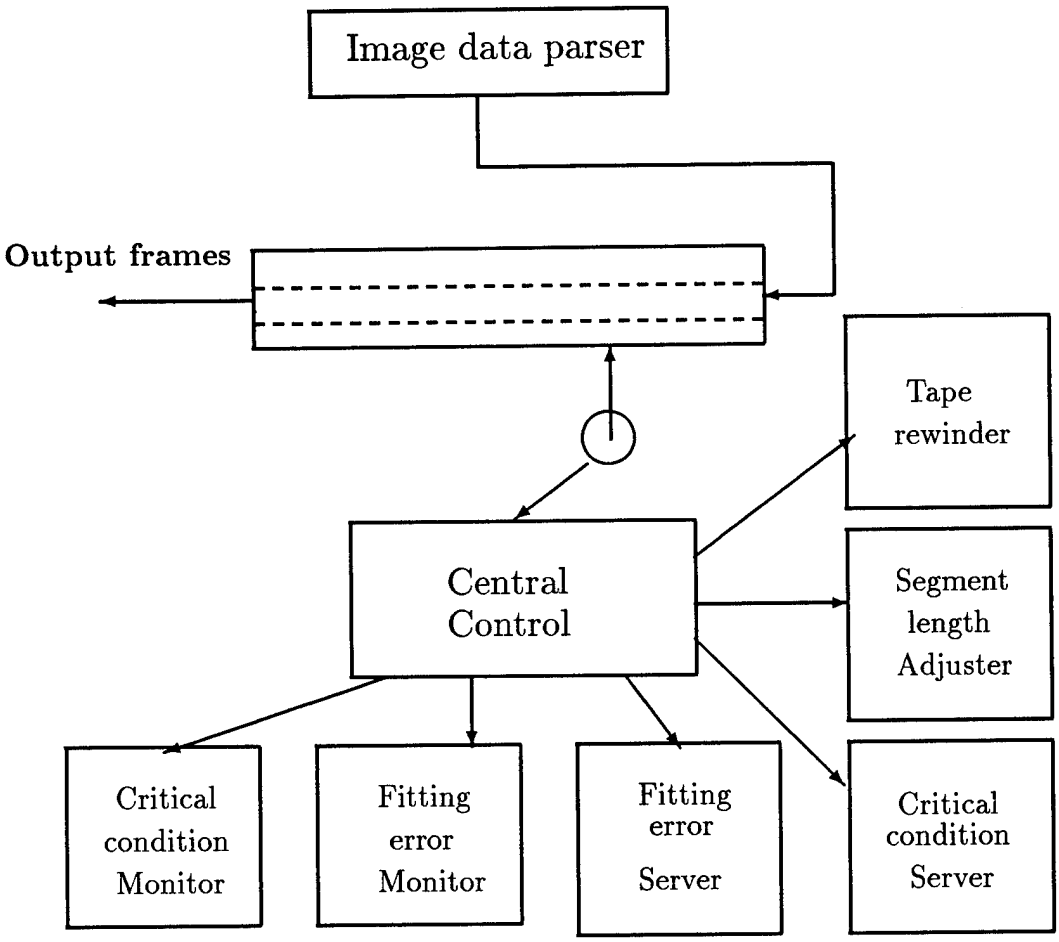

Figure 1: Modules in MPR

certain number of frames have been parsed. If a critical frame is found, the *Control* calls the *Tape rewinder* to take the pointer to the buffer to the critical frame. Because reconstructed motion is sensitive to the segment length of the figure model used to fit the image sequence, we need to decide whether an unusual pattern of motion is due to the discrepancy of the segment length between the fitting model and the observed person, and adjust the fitting model accordingly if so. This task is taken care of by *Segment length adjuster*. As is explained in my proposal, which I submitted in a previous quarterly report, there are two alternative interpretations following a critical frame. The module *Critical condition server* chooses a more reasonable direction for the reconstruction process. The next thing the *Central control* needs to take care of is the fitting error surge. It is detected by *Fitting error monitor*. If it spots one, the Control calls *Tape rewinder* to rewind the buffer pointer to the suspicious critical frame and calls *Fitting error server* to force the reconstruction to follow the alternative path. When it reaches the end of the buffer, the Control calls the parser to read in a new frame. The detailed description will be given in the next report.

I have tried some examples of one arm movement. The image data is derived from the 6-D data from Dr. Maulucci of MOCO. MOCO did many one-arm reach experiments, where the subject has 6-D sensors attached on her lower and upper arm. The frequency of their data is 100 Hz. I used the *Jack* human figure model to fit the reach data, and then generated image sequence from the model. Since the image sequence comes from the figure model we use to reconstruct the 3-D postures, we first tested our program on the situations where there is no modeling error or reconstruction error caused by the discrepancy of the original real human figure and the computer figure model used to reconstruct the 3-D postures. Our program tests well when there is no modeling error. It successfully detects the critical configurations and recovers from the ambiguities inherent to monocular viewing. It also adjusts the segment length when the upper arm of the fitting model is 5% shorter.

I have yet to test the program on motion involving two arms and the torso. I shall also work on how to recover when the model has a longer segment or, starting from a shorter segment, how to gradually adjust the segment lengths and converge to real lengths up to a perspective factor.

12 Virtual Reality: Rebecca Mercuri

My work during this period was concentrated in two areas, Virtual Reality and Visualization.

- Virtual Reality:

  Ranjit Bhatnagar, Brian Stokes and I worked on the implementation of "Sound World," a virtual environment where objects have affiliated sound properties. As the user moves around in the world, colliding with objects, their sounds are emitted. Combinational sounds are also possible.

  A number of virtual musical instruments were constructed by Brian Stokes and Angela Lei under my supervision. These include: Theremin, Autoharp, Drumset. The theremin uses

algorithmically produced sounds with variable pitch and volume controls (software by Ranjit Bhatnagar). The autoharp and drumset use digitized sound samples.

- Visualization:

  The *Jack* system is being modified to include an audio development menu that will allow the interactive attachment of algorithmic or digitized sounds to objects in the environment. This work is being executed by Teo Kok Hoon (see section following).

I have also been guiding Leanne Hwang, an undergraduate, in the construction of lighting models that are to be rendered using Min-Zhi Shao's radiosity software. These models will be of use in the audio visualization project, when they are extended to the sound domain.

Angela Lei constructed a number of sound icons and objects (microphone, loudspeaker, headphone, sinusoid signal, ...) which will be used in the audio visualization project. Angela and I were also working on a visualization model for radiating sounds, using algorithmically generated 3D objects (spheres, cartioids) displayed concentrically with varying color.

Some of the above work was demonstrated at the Institute for Music and Healing held in August at Immaculata College. The presentation was covered by the Philadelphia Inquirer, August 13, 1992, Main Line Neighbors Edition.

13 Audio *Jack*: Kok-Hoon Teo

The main aim of this project is to equip *Jack* with audio capabilities. The idea is to tag objects (figures, segments, sites, etc) with either a sound- generating function or a sampled audio file so that these associated forms of audio data can be used to illustrate changes in the object's orientation, position as well as other properties. Ultimately, we want to be able to localize the various audio sources using a pair of stereo headphones.

This new version of *Jack* has an additional "audio" menu. At this moment, only files containing audio samples in the AIFF format are supported. Only the filename is required of the user during the attachment process. Thereafter, the "delete" menu item allows the removal of the audio files while the "edit" menu allows the user to select a subset of all the available audio files to be played.

Currently, the audio triggering function "audio_control_play_all" calls up a shell script "play"" with the list of active audio files as its argument. This allows flexibility in deciding how the audio files should be played, whether simultaneously or one after another. The audio files are played to the audio output device using the "sfplay" program. As the sound chip only supports 4 separate sound ports, a maximum of 4 audio files can be played at one time.

It should be possible to rewrite certain routines in the animation system of *Jack* to play back any audio files attached to an object. This would be ideal for illustrating collisions of objects, or even footsteps of *Jack* as he wanders about.

Moreover, we want to be able to attach different audio triggering functions to objects as well, so that we can have sounds which vary continuously with certain properties of the object (eg. joint angle).

A collection of sampled sounds can be found in the /usr/lib/sounds/prosonus directory. These files are in AIFF format, typically 44.1 kHz sampling rate.

14 Optimal Mesh Algorithms for the Voronoi Diagram of Line Segments, Visibility Graphs and Motion Planning in the Plane: Suneeta Ramaswami

The motion planning problem for an object with two degrees of freedom moving in the plane can be stated as follows: Given a set of polygonal obstacles in the plane, and a two-dimensional mobile object B with two degrees of freedom, determine if it is possible to move B from a start position to a final position while avoiding the obstacles. If so, plan a path for such a motion. Techniques from computational geometry have been used to develop exact algorithms for this fundamental case of motion planning.

We are interested in studying special cases of algorithmic motion planning and the related geometric problems using parallelism. For a number of special cases of motion planning, we know that their sequential algorithms are either optimal or close to optimal because of known lower bounds. Hence, any speed-ups obtained from a single sequential processor will be limited by constant or small factors. Parallel algorithms, however, offer the possibility of significant speed-ups. Our research aims at obtaining optimal parallel algorithms for such motion planning problems and will be aided by the considerable progress that has been made in the area of parallel algorithms for computational geometry in recent years ([1, 2, 3], for example).

Our focus is currently on the development of such parallel algorithms on a particular fixed-connection parallel network architecture, namely the *mesh-connected computer.* The *mesh-connected computer* (*mesh*) of size n is a fixed-connection network of n simple *processing elements (PEs)* that are arranged in a $\sqrt{n} \times \sqrt{n}$ two-dimensional grid. Each PE is connected to its (at most) four nearest neighbors. Attractive features such as simple near-neighbor wiring and ease of scalability have made the mesh-connected computer the focus of considerable attention in parallel algorithms research.

We have obtained optimal mesh implementations of two different methods for planning motion in the plane. We do this by first developing optimal mesh algorithms for some geometric problems that, in addition to being important substeps in motion planning, have numerous independent applications in computational geometry. In particular, we can show that the *Voronoi diagram* of a set of n nonintersecting (except possibly at endpoints) *line segments* in the plane can be constructed in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ mesh, which is optimal for the mesh. Consequently, we obtain an optimal mesh implementation of the sequential motion planning method given by Ó'Dúnlaing

and Yap [4]; in other words, given a disc B and a polygonal obstacle set of size n, we can plan a path (if it exists) for the motion of B from a start position to a final position in $O(\sqrt{n})$ time on a mesh of size n. Next we show that given a set of n line segments and a point p, the set of segment endpoints that are visible from p can be computed in $O(\sqrt{n})$ mesh-optimal time on a $\sqrt{n} \times \sqrt{n}$ mesh. As a result, the *visibility graph* of a set of n line segments can be computed in $O(n)$ time on an $n \times n$ mesh. This result leads to an $O(n)$ algorithm on an $n \times n$ mesh for planning the shortest path motion between a start position and a final position for a convex object B (of constant size) moving among convex polygonal obstacles of total size n.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. K. Yap. Parallel Computational Geometry. *Algorithmica*, 3:293–327, 1988.

[2] M. J. Atallah, R. Cole, and M. T. Goodrich. Cascading Divide-and-Conquer: A Technique for Designing Parallel Algorithms. *SIAM J. Comput.*, 18(3):499–532, June 1989.

[3] C. S. Jeong and D. T. Lee. Parallel Geometric Algorithms on a Mesh-Connected Computer. *Algorithmica*, 5(2):155–177, 1990.

[4] C. Ó'Dúnlaing and C. K. Yap. A 'Retraction' Method for Planning the Motion of a Disc. *J. Algorithms*, 6:104–111, 1985.

15 Flock of Birds Interface To *Jack*: Mike Hollick

The **Flock of Birds** (referred to hereafter as the *FOB*) is a 6 degree of freedom position and orientation measuring system developed by Ascension Technology. The interface allows the user to use FOB sensors to control figures in the *Jack* environment. Commands allow figures to be bound to a sensor's position, orientation, or both, and translational scaling and offsets may be defined. By using bound figures as the goals of constraints, articulated figures may be easily manipulated.

15.1 Hardware/Operating System Information

Configuring your system to use the FOB interface is straightforward. *Jack* supports FOB systems with either the standard transmitter or the Extended Range Transmitter. Instructions for properly connecting the components of your Flock are provided by Ascension Technology.

Each control unit of the FOB must communicate with your computer through a separate RS232 cable, and should be set at a communications rate of 9600 baud. The *Jack* interface is currently limited to three sensors; this will be changing shortly to allow four or more. Permissions on the serial ports must be set properly to allow access to all users. To set the permissions on the first

four serial ports, issue this command as root: `chmod ugo+rwx /dev/ttyd[1-4]`. This allows global read/write access to these ports.

15.2 Using the FOB in *Jack*

15.2.1 Commands

Commands for using the FOB are found in the **utility** → **flock of birds** menu. Here is a brief description of each.

- **configure FOB** This sends the appropriate commands to configure the Flock. It must be issued before any other FOB commands. It will prompt you for the number of birds you would like to use. Note that this number may be smaller the the total number of Birds in your Flock.
- **set scale factor** You can set the translational scaling for each Bird with this command. It will prompt you for a sensor number and a real number that will be used as the scaling factor.
- **set bird hemisphere** Each Bird should be operated in one hemisphere of the transmitter (details in the FOB manual). Use this command to set the hemisphere for each Bird individually; the **forward** hemisphere is the default.
- **enter bird offsets** This command allows you to enter the translational offsets for each Bird. It will prompt you for a sensor number and an offset. Offset units are assumed to be centimeters.
- **match bird position to figure** Bird offsets may be automatically set with this command to match the sensor's current position with that of a figure in the *Jack* environment. The command will prompt you for a Bird number and a figure. It will then sample the sensor's position, and adjust its offsets to match it to the position of the selected figure.
- **reset bird offsets** After prompting you for a Bird number, this command will reset the offsets of that sensor to zero.
- **attach figure to bird** This command binds the selected figure to a Bird. After this command is executed, a simulation function is created that continuously samples the position and orientation of the selected sensor, and adjusts the figure to match it. Offsets and scaling are set by the above commands.
- **attach figure to bird position only** This is like the previous command, but the sensor's orientation is ignored.
- **attach figure to bird orientation only** The orientation of the selected figure is matched to the sensor; position is not altered.

- **unattach figure from bird** This command prompts you for a Bird number, and releases the figure in the environment that was bound to it.

15.2.2 Interface Details

Whenever a figure in the environment is bound to a Bird input, a simulation function is created that communicates with one sensor unit. At the end of each simulation cycle, a request is sent to the Bird for an update. This is done at the end of each cycle for performance reasons. Due to the latency imposed by the operating system, there is a significant delay between when the bird data arrives at the port and when it is available to *Jack*. Thus, the strategy is to request an update at the end of the cycle, and read it at the beginning of the next, when it should be available. This imposes a one frame delay on the data, however.

Jack will attempt to restart the Flock if it goes offline for some reason. If there is a communication timeout, a message will be printed to `stderr`, along with a beep. It is necessary to power cycle the entire flock at this point. Since it takes a second or two for the control units to initialize themselves, they must be power cycled at the correct time. After the Flock goes offline, power each one down. Then, immediately after one of the beeps (*Jack* will continue to send error messages every few seconds until the Flock is communicating again), turn all the control units back on. After the next beep the Flock should restart. One important note- at times a timeout may occur while the Flock is still operational. This will be seen as a pause in *Jack*'s operation, followed by an error message. This can be safely ignored, as *Jack* will recover from this error without the need for power cycling the Flock.

All configuration information for each Bird is stored by *Jack*. When an error condition occurs, the Flock is re-initialized, then each configuration command is sent again to each bird. This is why it is necessary to turn the Birds on immediately after abeep. By doing this, you insure that a control unit will not come online in the middle of a message and miss some configuration commands.

15.2.3 Tips for Use

The FOB is not currently integrated into *Jack*'s constraint system. Thus, it is necessary to be just a little tricky if you'd like to use a Bird as the goal of a constraint. The easiest way to do this is to read a small object into the environment that will represent the Bird. Bind this object to the appropriate sensor, and make the base site of that object the goal of your constraint. Then, you can issue the **turn segment off** command, pick that object, and it will no longer be displayed. The goal of the constraint will then be matched to the position/orientation of the sensor.

Orientation offsets are not currently supported by the interface. This should be changing shortly, but in the meantime there is a way to work around this deficiency. If you would like to use a Bird as the goal of a constraint with the orientation offsetted, you can create a small object that will be used to represent the Bird in the environment. Create a site on the object that is offsetted by

the correct amount from the object's base site (see the *Jack User's Guide* for more information). Then, attach this figure to the Bird, and make the offsetted site the goal of the constraint.

15.3 Future Improvements

There are several areas in which the FOB interface will be improving in the future:

- Changing the communication procedure to allow a greater number of Birds in the Flock.
- Integrating Bird input with the constraint system, allowing sensors to be the goals of constraints directly.
- Adding orientation offsets to the system.
- Allowing Bird input to be used in direct manipulation. This would allow the user to use a Bird to move figures, adjust joints, change view, etc.
- Using the FOB to record motions. These motions could then be used in conjunction with *Jack*'s animation system.