9-1978

The Simulation of Human Movement by Computer

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Joseph O'Rourke

Stephen W. Smoliar

Lynne Weber

The Simulation of Human Movement by Computer

Abstract

This paper is concerned with a software simulation of movement of the human body. This simulation is being designed to drive a system for computer animation as part of a larger program concerned with the translation of movement notation into animated graphics. The simulation is based on a model of the human body as a network of special-purpose processors -- one processor situated at each joint of the body -- each with an instruction set designed around a set of "primitive movement concepts." We shall discuss the extent to which all these processors may employ the same architecture and the function of the network structure.

Disciplines

Computer Engineering | Computer Sciences

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report.

The Simulation of Human Movement by Computer

Movement Project Report No. 14

Norman I. Badler
Joseph O'Rourke
Stephen W. Smoliar
Lynne Weber

September 1978

Work reported herein was supported by a grant from the National Science Foundation, Grant No. MCS76-19464.

The Simulation of Human Movement by Computer

Movement Project Report No. 14

Norman I. Badler
Joseph O'Rourke
Stephen W. Smoliar
Lynne Weber

September 1978

Work reported herein was supported by a grant from the National Science Foundation, Grant No. MCS76-19464.

The Simulation of Human Movement by Computer

Norman I. Badler*
Department of Computer and Information Science
The Moore School of Electrical Engineering/D2
University of Pennsylvania
Philadelphia, PA 19104

Joseph O'Rourke
Department of Computer and Information Science
The Moore School of Electrical Engineering/D2
University of Pennsylvania
Philadelphia, PA 19104

Stephen W. Smoliar
General Research Corporation
P.O. Box 6770
Santa Barbara, CA 93111

Lynne Weber
Peat, Marwick, Mitchell, & Co.
345 Park Ave.
New York, NY 10022

September 1978

---

* Author to whom correspondence should be addressed.

The Simulation of Human Movement by Computer

Norman I. Badler
Joseph O'Rourke
Stephen W. Smoliar
Lynne Weber

Abstract

This paper is concerned with a software simulation of movement of the human body. This simulation is being designed to drive a system for computer animation as part of a larger program concerned with the translation of movement notation into animated graphics. The simulation is based on a model of the human body as a network of special-purpose processors -- one processor situated at each joint of the body -- each with an instruction set designed around a set of "primitive movement concepts." We shall discuss the extent to which all these processors may employ the same architecture and the function of the network structure.

The Simulation of Human Movement by Computer

Table of Contents

List of Figures

1. INTRODUCTION

We wish to enable a digital computer to translate movement notation into an animated display of human figures performing the represented movements. This process involves analyzing the descriptive content of movement notation systems, evaluating structures for the animation of human movement, and formulating a suitably realistic human body model [3]. The fundamental premise involves modelling the human body as a network of special-purpose processors -- one processor situated at each joint of the body -- each with an instruction set designed around a set of "primitive movement concepts." The translation of notation into animated movement may then be divided into two stages: a compilation stage in which the movement notation gets translated into programs for these special-purpose processors, and a simulation stage which simulates the behavior of these processors as they interpret their respective programs [21].

We are currently working with Labanotation [12], a movement notation system chosen for its logic, its flexibility, and the extensive amount of material recorded in it. As a result of research into the development of a text editor, we have established a structured description of Labanotation text in terms of graphic primitives [20]. What we require for an instruction set for our special-purpose processors is a structured description of the same information in terms of movement primitives. While the structure of this instruction set has been influenced by the semantics of the graphic primitives of Labanotation, it is sufficiently general to be used to represent information recorded by other movement notations. For purposes of this discussion, we shall regard each instruction type as a data structure and describe it using the notation of Hoare [11]. A sequence of instructions associated with a joint processor specifies the movement of

that joint over a period of time. Thus, the compilation of Labanotation text produces a set of sequences of instructions, one sequence for each joint whose movement is notated.

Alternative methods for specifying movements may be found in techniques for computer animation. A three-dimensional model of the human body is constructed, articulated at joints and movable in space [2]. A number of techniques for describing the movements of such a model are reviewed elsewhere [3]; the method we have chosen is to simulate the movements of each joint of the body. The simulation bears some resemblance to the control structures for robot manipulators incorporated in AL [10] and LAMA [15]. These systems are primarily goal-directed and maintain an internal model of the device and its environment. Constraints such as those used by Spegel [22] are also used to control joint movements when external contact surfaces are involved in movements affecting support. Other graphic languages for animation offer subsets of the set of instructions we shall describe next.

2. THE ARCHITECTURE OF THE SIMULATOR

The abstracted movement concepts fall into five categories: 1) direction signs, 2) revolution signs, 3) facing signs, 4) contact signs, and 5) shape descriptions. Direction signs include those symbols which essentially describe the translation of some joint of the body, while revolution signs allow for the description of various forms of rotational movement, such as turning, twisting, and pivoting. Facing signs involve the establishment of an orientation, which is generally accomplished through a joint translation, a rotation, or a combination of the two. Contact signs indicate contact of body parts with other body parts, other people, the floor, the performer's clothing, or other physical objects. Shape descriptions are used to describe the tracing of a path or formation of a shape by some part of the body.

Each category stems from a different way of analyzing motion. Different movement notation systems tend to concentrate more heavily on one type of analysis over another. Iconographic systems, which are essentially based on stick figures, such as Jay notation [13], Sutton notation [18], and, to some extent, Benesh notation, [4], rely heavily on shape descriptions. Labanotation tends to use direction signs most heavily, while the system developed by Eshkol and Wachmann [8] concentrates on rotations and circular movements. Animation languages usually offer geometric transformations, such as rotational movements, and shape descriptions [7,22]. Manipulator langauges include limited contact specifications [10,15]. A system to describe movements of rigid or articulated objects permitted concepts related to direction, revolution, and contacts [1].

The instruction set discussed herein should be adequate for the

representation of any notation or animation system currently used. Before we consider the structure of each category in detail, let us first discuss how the selection of the individual joint processors relates to the structure of the human body.

2.1. A Model of Body Structure

Figure 2.1 [12] presents a "first approximation" of an assignment of processors to body joints. In this illustration each joint is labeled by the Labanotation symbol which represents it. (We have generalized the term "joint" to include body extremities.) This assignment of processors may, if necessary, be further refined for greater detail. However, for the purposes of the discussion in this paper, the detail in Figure 2.1 is sufficient.

Each joint processor positions its associated joint with respect to a cross of axes which defines a rectangular coordinate system. This cross of axes is generally situated at a second joint of the body. For example, movement of the right lower arm is determined by the processor at the right wrist with respect to a cross of axes situated at the right elbow. Alternatively, movement of the entire right arm is determined by the same processor at the right wrist but with respect to a cross of axes situated at the right shoulder. In describing any movement, the distal joint is defined to be the joint at which the active processor is located; and the proximal joint is the joint from which movement is effected. (In the above two examples the proximal joint is also the location of the cross of axes.) The term "body part" will be used to refer to a portion of the body which lies between a given proximal joint and a given distal joint.

All joint processors are essentially "computationally independent." All information exchanged among processors is transferred through a common monitor.

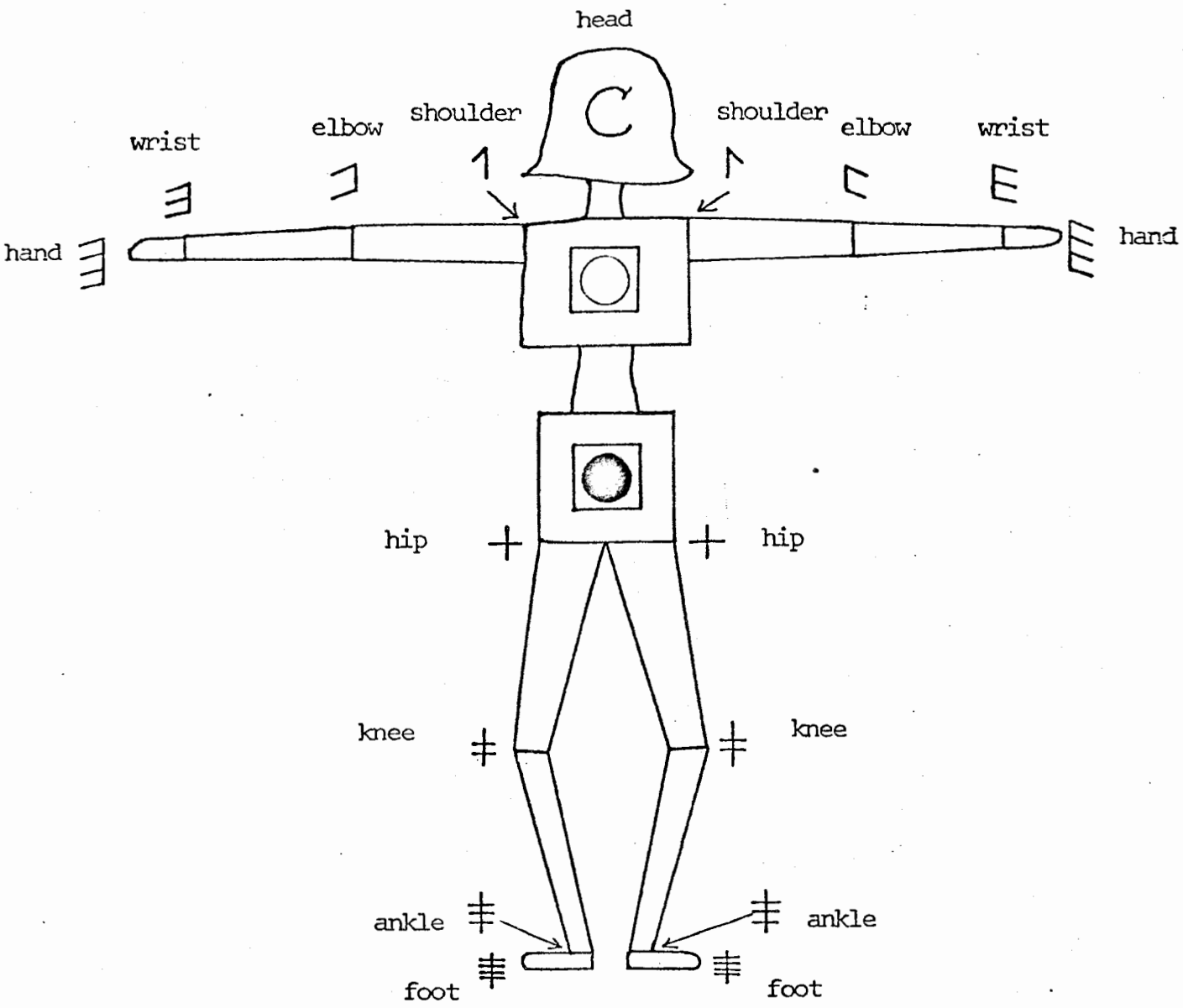


Figure 2.1 Major body joints.

One of the major responsibilities of this monitor is the maintenance of a data base regarding body position. Through this data base any joint processor may obtain current information on all joint positions. Since these positions may be specified with respect to an arbitrary cross of axes, the monitor is responsible for conversion of positional information between the various systems of reference. In addition, the monitor determines the timing and sequencing of command execution by scheduling individual joint processors to minimize conflicts between their actions. Finally, the monitor assumes primary responsibility for the interpretation of contact signs.

In addition to the classification of movements according to the instruction categories given above, a movement may also be described as either a gesture or a support movement. A support is a movement of the body's center of gravity. (The center of gravity may be slightly displaced as a result of gestural movement; but in a gesture, displacement of the center of gravity is an effect, while in a support movement its displacement is the cause of the movement.) Support movements are implemented by a progression processor capable of dispatching commands to any joint processors involved in locomotion. The progression processor is also responsible for maintaining the body's balance (i.e. the center of gravity over the base of support). Now let us discuss the actual instructions executed by the joint processors.

2.2. Instruction Structure

2.2.1. Direction signs

A direction sign specifies the translation of a joint as either a position description or a movement description. The former describes the

orientation of a distal joint with respect to a cross of axes, while the latter describes a path of motion with respect to the initial position of the joint. The necessary components of a direction sign are duration (given in terms of a simulation time unit), direction, designation of proximal joint, and the specification of either position or movement description. Optional components allow for modification of the path of motion, which may involve the movement of other joints. This may be summarized by the following type declarations:

```
type direction signs = (duration:  rational;
                        direction:  direction description;
                        proximal joint:  joint name;
                        kind:  (position, movement);
                        modifiers:  powerset direction modifiers).

type direction modifiers = (placement modifier,
                            sequence deviation description,
                            sequence inclusion,
                            sequence intermediate joints).
```

A brief explanation of this data structure notation is in order. The semicolon is a delimiter of components of a data structure, all of which must be present. The name of each component is given to the left of the colon. The comma is a delimiter of elements of a set. Normally, this is interpreted as a list of alternatives, exactly one of which is present. (Thus, the "kind" component contains either the element "position" or the element "movement".) Powerset indicates a subset of any size (including the empty set) of its argument; and sequence indicates a sequence of any length (including the empty sequence) of elements of a designated set.

The structure of a "direction description" involves further detail:

```
type direction description = (normal:  coordinate-triple;
                              modified:  modified coordinate-triple).
```

```
type coordinate-triple = (horizontal:  (right, place, left);
                          level:  (low, middle, high);
                          saggital:  (backward, place, forward)).

type modified coordinate-triple = (head:  (normal:  coordinate-triple,
                                           null:  nil);
                                   tail:  sequence orientation mods).
```

A normal coordinate-triple is capable of specifying one of 27 possible orientations with respect to a cross of axes. These are illustrated in Figure 2.2 [14]. The cross of axes is situated in the center of the middle plane, with the "forward" direction pointing into the page and the "right" direction pointing to the right. (The symbols at each of the points give the representation of each triple in Labanotation.) The actual distance to these points is determined by the length of the body part being moved.

Any other orientation is specified by applying a list of modifiers to a given coordinate-triple. (If the head of a modified coordinate-triple is nil, the modifiers are taken to apply to the current orientation of the processor.) There are two classes of orientation modifiers. An angular modifier specifies a direction with greater precision than a normal coordinate-triple, and a radial modifier specifies an alteration in the distance along the given orientation:

```
type orientation mods = (angular:  (amount:  fraction;
                                    direction:  coordinate-triple),
                         radial:  bend/stretch).
```

An angular modifier is given by a fraction and a coordinate-triple. For example, the triple (place; middle; forward) may be modified by (½; (place; high; forward)) to indicate a direction halfway between middle-level forward and high-level forward. (This is illustrated in Figure 2.3 [12].)

One might ask why we have not chosen a direction description to be represented simply by an ordered triple of real numbers. The reason is that we are interested in the simulation of the movement behavior of the human body.

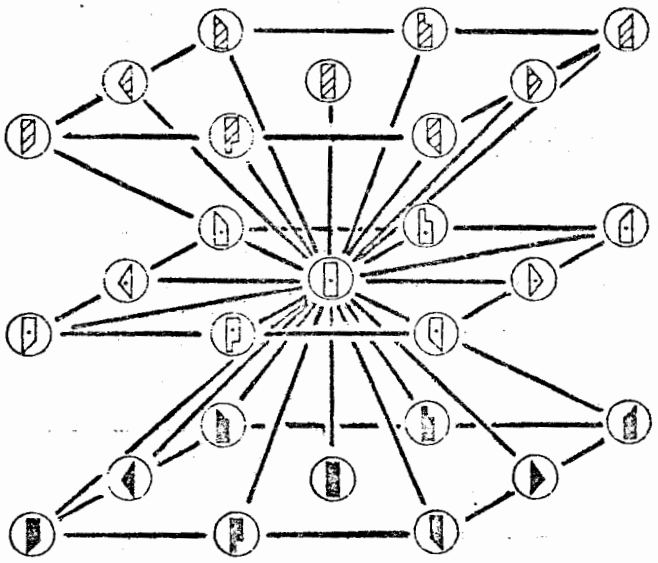


Figure 2.2 The orientations represented by a coordinate triple.

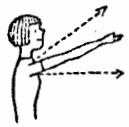


Figure 2.3 Angular modification of orientation.

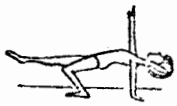


Figure 2.4 Demonstration of alternative systems of reference.

Arm alone

Upper body included

Figure 2.5 Inclusion.

While the body is quite poor at picking out arbitrary points in space, it is fairly good at establishing the direction of "right" or "forward." We have tried to follow a philosophy that those orientations which are more difficult for the body to determine should be reflected by more complex expressions in the instruction set.

Radial modification arises from the ways a body part may be bent or stretched:

```
type bend/stretch = (kind:  bend/stretch description;
                     direction:  surface).

type bend/stretch description = (kind:  (bend,
                                         stretch,
                                         fold,
                                         unfold);
                                 degree:  fraction).
```

The distinguishing features of bending, stretching, folding, and unfolding have been discussed by Hutchinson [12]. Each specification is quantified by a fraction. For example, the arm may be described as being 2/3 of the way from unbent to fully bent. (A degree of zero would indicate the unbending of a bent arm; this is sometimes called neutralization.) The direction of a bending or folding movement is designated by a point on the surface of the body which becomes "covered" by the movement; stretching and unfolding take their direction from the complementary movement of "uncovering."

Now let us consider the direction modifiers. Modification of the cross of axes involves specification of its origin and orientation:

```
type placement modifier = (origin:  (internal:  (locus:  (joint name, surface),
                                                 general:  whole body),
                                    external:  (locus:  direction description,
                                                 general:  whole room));
                          orientation:  (right-left:  orientation desc;
                                         low-high:  orientation desc;
                                         backward-forward:  orientation desc)).
```

The origin of the cross of axes may be located either on the body or in the external environment. In the former case, it may be located at any of the joints enumerated in Figure 2.1:

```
type joint name = (side-of-body:  (right, center, left);
                   area:  (clavicle,
                           shoulder,
                           elbow,
                           wrist,
                           end of hand,
                           hip,
                           knee,
                           ankle,
                           end of foot,
                           upper rim of pelvis,
                           lower rim of rib cage,
                           neck,
                           head)).
```

"Center" is used to describe the "side-of-body" of joints which do not come in pairs. Also, "center" is used with "shoulder" and "hip" to indicate the upper chest and lower pelvis, respectively. The cross of axes may also be located at a point on the body surface (using a technique described below). When "whole body" is specified as the origin, it is not fixed at a given locus; and only the orientation information may be interpreted. Similarly, external orientation may be given as a point in the room (by a direction description) or by a "whole room" designation.

Orientation is specified by giving a direction for each of the three coordinate axes. (Only two of these axes need be specified; the third is the perpendicular to the other two which forms a right-handed coordinate system.) Orientation may be defined by two points in space, two points on

the body, the perpendicular to a body surface, or the tangent to the current line of direction:

```
type orientation desc = (space:  (direction description;
                                  direction description),
                         body:  (joint name; joint name),
                         perpendicular:  surface,
                         line of direction:  tangent flag).
```

Figure 2.4 [12] illustrates a variety of orientations which could have several possible descriptions. Consider, as an example, the right arm. To describe the position of the right arm, it is necessary to locate the right wrist with respect to a cross of axes whose origin is at the right shoulder. There are at least two ways in which the axes themselves may be oriented. One alternative is to align the "low-high" axis with gravity. This would entail the "space" description: ((place; low; place); (place; high; place)). (Imagine Figure 2.2 as a schema for selecting points in space.) The other possibility is to align the "low-high" axis with the torso. This may be achieved by a "body" description: ((center; hip); (center; shoulder)). In both cases the "right-left" axis may be given by the "body" description ((right; shoulder); (left; shoulder)); and the "backward-forward" axis is the perpendicular to the plane described by the other two axes. (A more detailed discussion of orientation alternatives is given in [9].) Of course, different coordinate-triples are required to describe the direction with respect to the alternative systems of reference. The former cross of axes requires the coordinate-triple (place; high; place); the latter requires (place; middle; forward).

There remains the specification of a point on the surface of the body. This may be determined as follows:

1) Select a body part by specifying its proximal and distal joints.

2) Select a point along that body part as a fraction of the total distance from the proximal joint to the distal joint.

3) Select a direction to proceed from the "bone" of that body part to the "skin" of the surface; this may be given as a fraction of full rotation from a "zero" position which is defined for each body part.

We thus have the following data structure:

```
type surface = (body part:  segment;
                displacement:  fraction;
                point:  fraction).

type segment = (proximal:  joint name;
                distal:  joint name).
```

This description technique may be demonstrated by giving an explicit definition of the "backward-forward" axis for the second alternative given above (in which "low-high" and "right-left" were both given by "body" descriptions). Intuitively, the direction is the perpendicular to the front chest surface. This corresponds to the "surface" descriptions: (((right; shoulder); (left; shoulder)); ½; 0). The "½" indicates the point halfway between the two shoulders, and "0" indicates the amount of rotation to face the front of the chest.

We conclude this section with a brief description of the remaining direction modifiers. A deviation is a slight directional displacement from the unmodified direction [12]. It is specified by the amount of time it endures and by the direction of deviation:

```
type deviation description = (duration:  rational;
                              direction:  direction description;
                              degree:  (greater, normal, lesser)).
```

Several deviations may occur in sequence during the execution of a direction sign.

An inclusion specifies the participation of other joints in a movement.

Figure 2.5 [12] illustrates the effect of including the upper body in a movement of the right arm. Describing an inclusion requires the following structure:

```
type inclusion = (delay:  rational;
                  part list:  powerset joint name;
                  degree:  (greater, normal, lesser)).
```

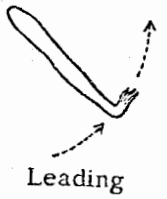
The components are the time of inclusion (given as a delay following the start of the movement being modified), the joints involved in the inclusion, and the degree of inclusion. Other movements which directly involve other processors are as follows:

```
type intermediate joints = (delay:  rational;
                            kind:  (leading:  powerset (joint name, surface),
                                    following:  powerset (joint name, surface),
                                    outward:  (coordinate-triple, nil),
                                    inward:  (coordinate-triple, nil),
                                    guidance:  surface)).
```

A given joint or a surface may be designated as leading the movement or following the movement of the whole (Figure 2.6 [12]). A movement may cause a sequential use of body parts outward (proximal to distal) or inward (distal to proximal) (Figure 2.7 [12]). This sequential use may or may not be modified by a direction specified by a coordinate-triple. Finally, a movement may be guided by a particular surface [12].

2.2.2. Revolution signs

Revolution signs specify movement about some axis. Consequently, the instruction must designate a duration, a proximal joint, an axis about which revolution occurs, the amount of revolution, and a descriptor to differentiate between twist and rotation. A twist is a revolution of a body part where the proximal end does not move to the same extent as the distal end. For example, the lower arm's natural movement about its central axis is a twist.



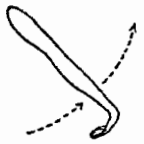


Figure 2.6 Leading and following.

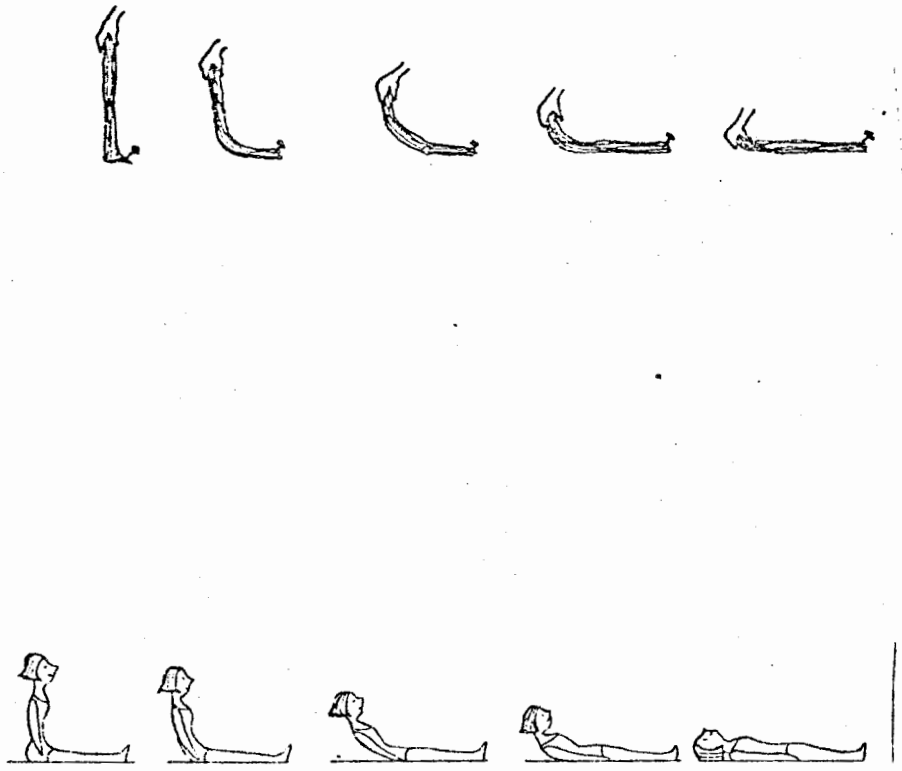


Figure 2.7 Sequential use of body parts.

A rotation indicates that the body part will turn uniformly. This is seen when the body as a whole turns. Hutchinson [12] discusses this distinction at greater length. A modifier may be present to allow for alteration of the cross of axes used to determine the axis of rotation:

```
type revolution signs = (duration: rational;
                         proximal joint: joint name;
                         axis: (direction description, segment);
                         amount: revolution quantity;
                         kind: (twist, rotate);
                         modifier: (placement modifier, nil)).

type revolution quantity = (sign: (clockwise, counterclockwise);
                            magnitude: rational;
                            origin: (current, stance, absolute)).
```

The "amount" component is given in sign-magnitude form, where the "sign" specifies whether the revolution is clockwise or counterclockwise and the "magnitude" is a rational number of full revolutions. In addition, an "origin" component specifies whether "amount" is measured from the current orientation of the body, the orientation of a front-facing stance, or an absolute front orientation associated with the external environment.

2.2.3. Facing signs

A facing sign indicates a relationship between a body part surface and a direction. This relationship may be expressed in terms of direction signs and rotation signs, but it entails a different approach to movement analysis. Like the other instructions, a facing sign requires a duration. Then the body part surface and the direction it faces are given. Finally, a modifier may be present to allow for alteration of the cross of axes used to determine the facing direction:

```
type facing signs = (duration: rational;
                     area: surface;
                     direction: direction description;
                     modifier: (placement modifier, nil)).
```

2.2.4. Contact signs

Direction signs, revolution signs, and facing signs are all interpreted in terms of a movement originated by a single joint. Contact signs, on the other hand, are interpreted by the monitor which supervises the behavior of all joint processors. A contact sign is represented as follows:

```
type contact signs = (time:  rational;
                      contacts:  sequence (object; relation)).

type object = (human:  (indicator:  person;
                        place:  surface),
               other:  auxiliary object description).

type relation = (kind:  (relate,
                         near,
                         touch,
                         support,
                         passive);
                 modifiers:  powerset contact modifiers).

type contact modifiers = (in passing, maintain, slide, surround).
```

The first component of a contact sign specifies the time at which the contact takes place. The next component specifies a list of contacts as object-relation pairs. If the "object" is a person, that person must be identified, together with the specification of a location on the body. The relation classifies the contact as a relationship (mutual "addressing" [12] among the objects involved, without actual physical contact), nearness (also does not involve actual physical contact), touch, bearing of weight, or a passive approach to the relation. Optional modifiers may specify the contact taking place "in passing," maintaining the relation, sliding after contact has been established, or surrounding the contacted object. (Cancellation of surrounding, maintained, or sliding contact may be represented by an instruction at a later time without the appropriate modifier.)

2.2.5. Shape descriptions

Shape descriptions specify paths or configurations of body parts. (At a higher level they may also be used to describe shapes of groups of people.) They require a duration, designation of a proximal joint, a plot of points in three-space to determine the shape, and a designation of whether the shape indicates a position or movement description. A modifier may be present to alter the specification of the cross of axes:

```
type shape descriptions = (duration: rational;
                           proximal joint: joint name;
                           path: sequence vector;
                           kind: (position, movement);
                           modifier: (placement modifier, nil)).

type vector = (x: real; y: real; z: real).
```

2.3. An Overview of Instruction Interpretation

The primary purpose of the compilation stage of our system, as cited in Section 1, is the preparation of a set of disjoint programs for the individual joint processors, the monitor, and the progression processor. Figure 2.8 shows how these processors are organized for the simulation stage, during which their programs are interpreted. The monitor is responsible for synthesizing a program which will then be passed on to a graphic processor. All contact signs are collected together in a single program for the monitor. Instructions affecting support are sent to the progression processor, and all remaining instructions are placed in the instruction streams of their respective joint processors. Within the progression processor instruction stream, support instructions are further partitioned: each block accounting for the movement of a single supporting joint. This partitioning is handled by the compilation stage since no instruction, in itself, indicates which joint processor it is meant to direct.

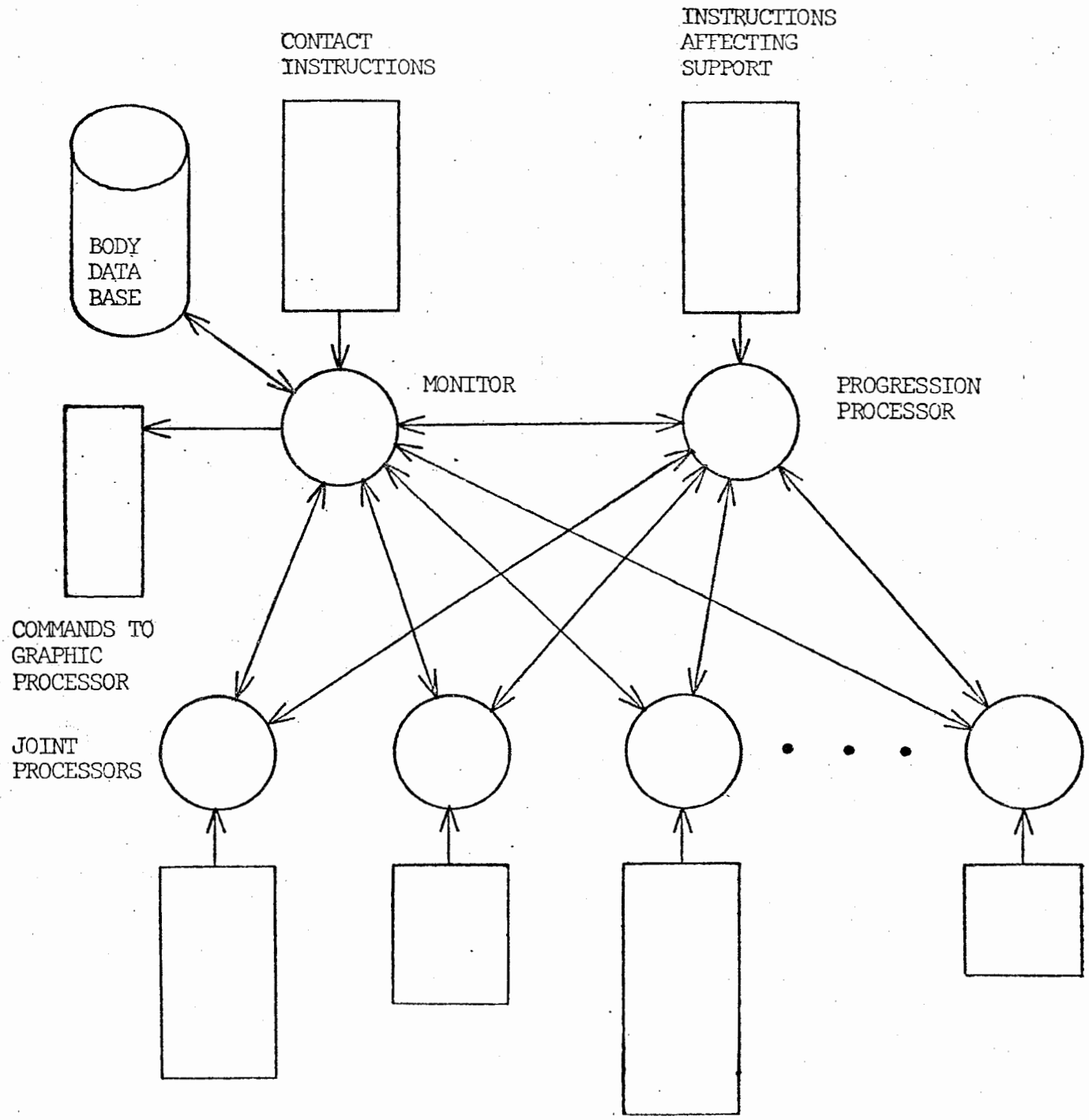


Figure 2.8 Programs for the system network.

All timing information for a joint processor is provided by the duration fields of its instructions. However, block-structured parallelism enables the representation of concurrent execution of several instructions by a single processor [9]. Thus, the absence of movement must be explicitly represented by a "null" instruction (analogous to a rest in music notation [19]); and all substreams of a concurrency block must fill the same duration interval. The structure of a possible instruction stream is illustrated in Figure 2.9. All instructions express time in terms of a rational number of time units. This unit is related to the simulation process by defining a simulation interval to be the real time between successive "snapshots" of the human figure desired by the graphic processor. The simulation interval is represented as a nonzero number, where the only restriction is that no instruction may begin or end between simulator "snapshots" or movie frames. The simulation interval may be fixed by the user or may be computed by the monitor based on the earliest starting time of the upcoming set of instructions. (Each processor can supply this information to the monitor.) By permitting the interval to vary, the simulator can treat quiescent periods more efficiently.

The general control of the simulator involves the iterative execution of the following six steps for each simulation interval:

1. The monitor generates instructions to initiate contacts.
2. All current activities are represented by (joint processor, instruction) pairs; these pairs are assigned a priority ordering based on body structure.
3. The monitor allows the progression processor to implement any currently active support movements.

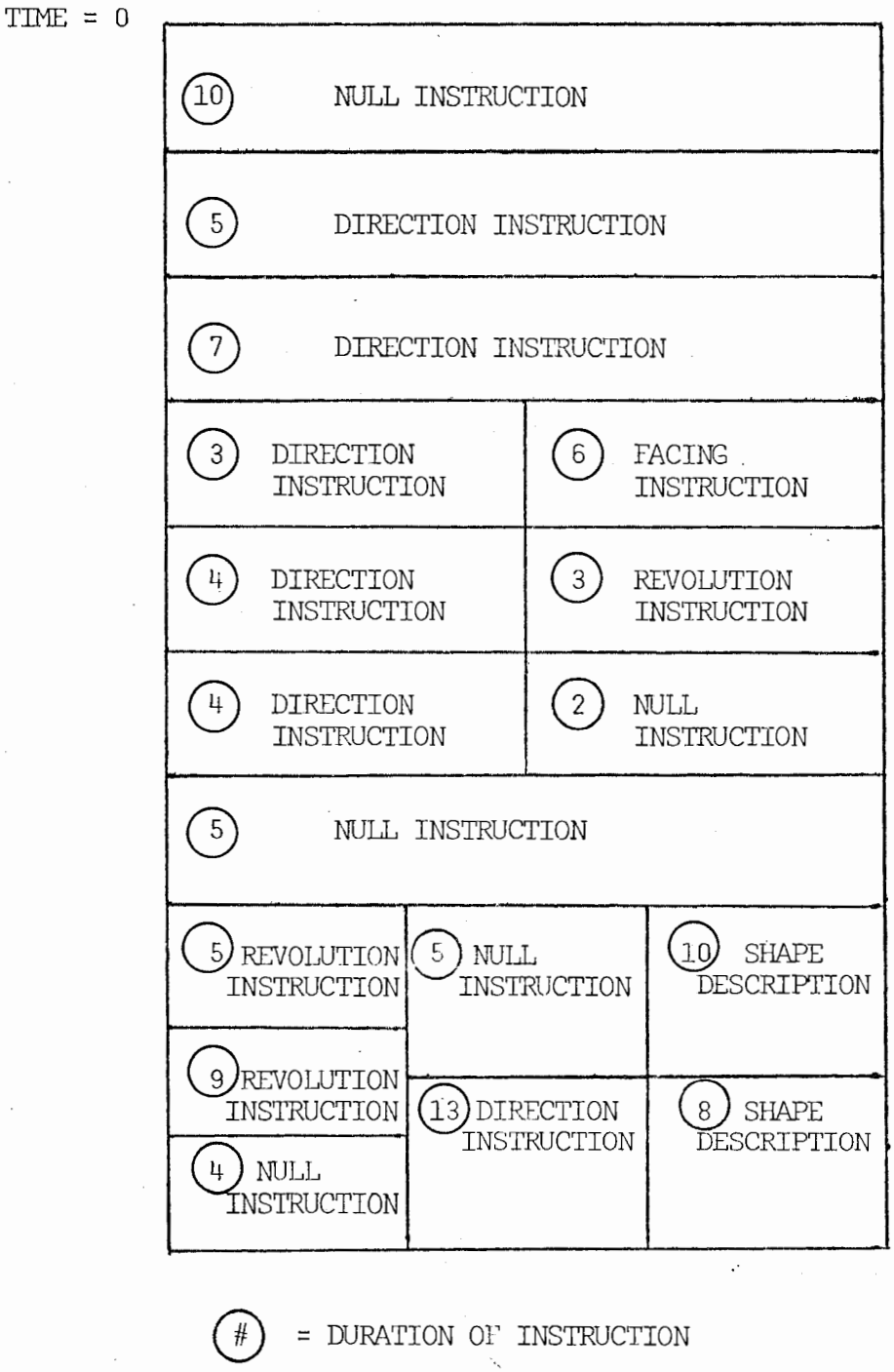


Figure 2.9 Structure of an instruction stream.

4. The monitor allows the implementation of each (joint processor, instruction) pair according to the priority order established in Step 2. Pairs with the same priority (and therefore the same joint processor) are executed concurrently by the joint processor.
5. The progression processor adjusts balance, if necessary.
6. The monitor calculates all final body positions and prepares the output for the graphic processor.

For each simulation interval, the monitor must first establish how contact instructions may be executed. Since contacts have no explicit duration prior to achievement, the monitor must utilize suitable existing instructions or synthesize new ones for appropriate joint processors. The monitor must next organize the actual execution of the other processes so that their sequential execution will in fact appear logically parallel. A priority ordering is computed by the monitor to insure an overall determinism to joint movements and to facilitate the manipulation of joint location information stored in the body data base (Section 3.2). Since a joint processor may be executing several instructions in a conceptually parallel fashion, a priority is computed for each active instruction of each processor. Should these priorities be the same, the processor itself establishes the order of execution within the set. The monitor now transfers control to the progression processor, signalling that modifications to the body data base may be performed. After support movements, the joint processors execute their instructions based on the order scheduled by the monitor. When all the joint processors have completed their current instructions for this cycle, the monitor again calls upon the progression processor to balance the body, if necessary.

At the completion of all processing for a simulation interval, the monitor outputs a single stream of commands to a graphic processor which constructs and displays an image of a human figure [2]. Over a sequence of simulation cycles an animation is produced. If desired, the monitor also generates a textual report of the model's position, contacts, and collisions.

2.4. An Example

As an example of the simulation process, consider the interpretation of the Labanotation segment illustrated in Figure 2.10. It describes one cycle of a normal forward walk: first the left foot steps forward, then the right foot follows. Because the direction signs for support actually describe the transference of weight, the feet do not move with respect to the floor during a forward direction sign; rather, the center of gravity moves forward [12]. During this forward movement of the center of gravity, the right arm first moves so as to point straight down from the shoulder, then moves to a position slightly forward of the body. The left arm moves in a complementary fashion. The final arm positions are shown in Figure 2.11.

The Labanotation segment is compiled into three instruction streams: two direction signs for the left wrist, two more for the right wrist, and two direction signs for the progression processor:

left wrist processor:

```
1.  (1;
     (place; low; forward);
     (left; shoulder);
     position;
     (((left; shoulder);
       (((right; shoulder); (left; shoulder));
        ((place; low; place); (place; high; place));
       (((right; shoulder); (left; shoulder)), ½; 0)))))
```

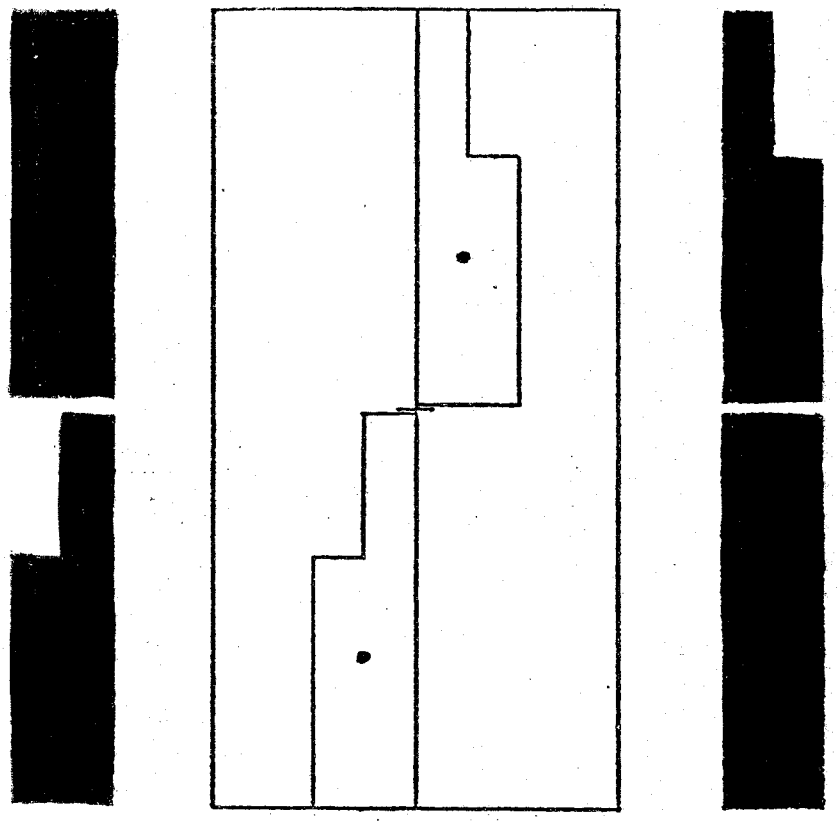
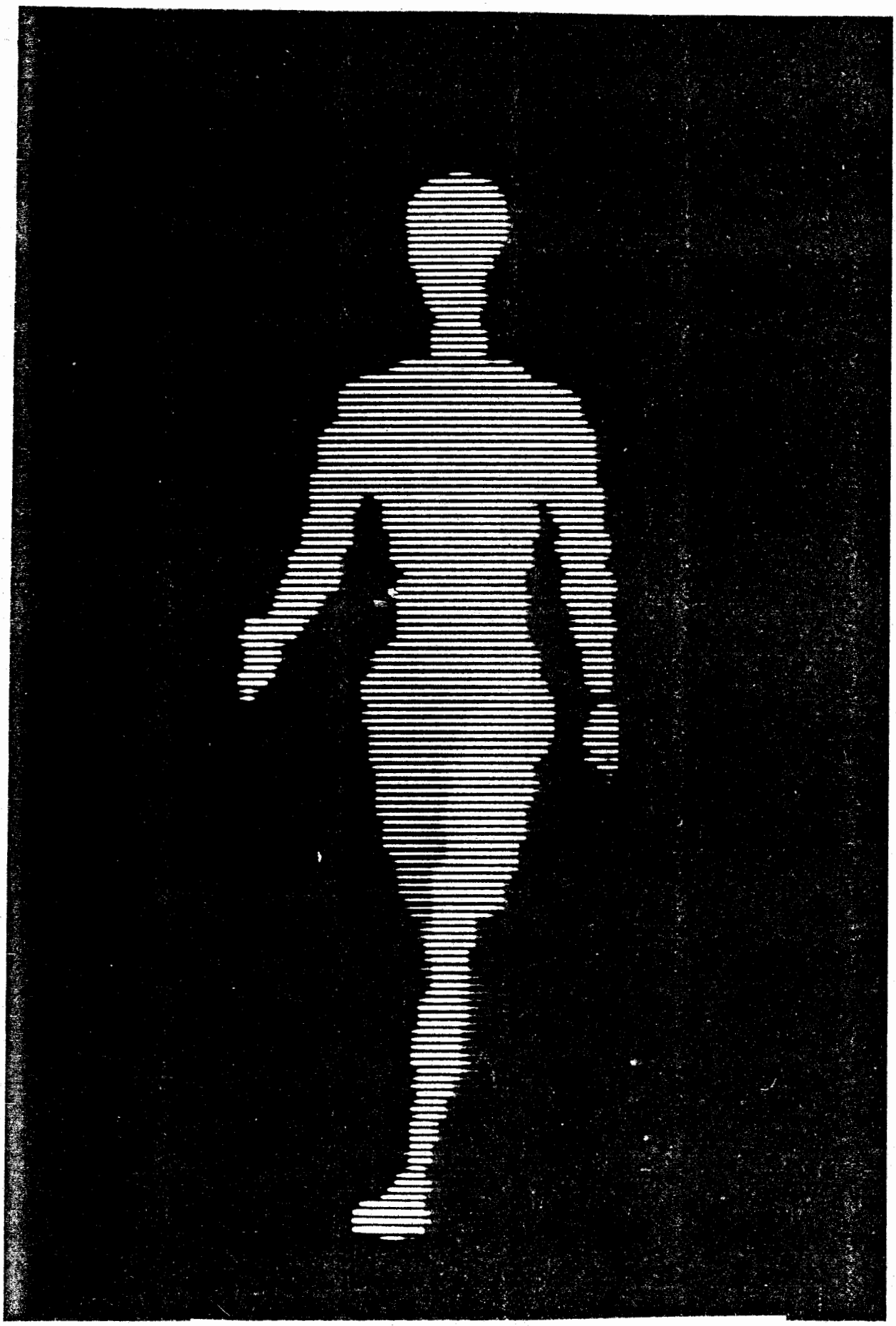


Figure 2.10 Labanotation segment for a simple walk.



```
2.  (1;
     (place; low; place);
     (left; shoulder);
     position;
     nil)
```

right wrist processor:

```
1.  (1;
     (place; low; place);
     (right; shoulder);
     position;
     (((right; shoulder);
       (((right; shoulder); (left; shoulder));
        ((place; low; place); (place; high; place));
        (((right; shoulder); (left; shoulder)); ½; 0)))))

2.  (1;
     (place; low; forward);
     (right; shoulder);
     position;
     nil)
```

progression processor:

```
1.  ((left; ankle);
     0;
     (1;
      (place; middle; forward);
      (left; hip);
     movement;
     ((whole body;
       (((right; hip); (left; hip));
        ((place; low; place); (place; high; place));
        (((right; hip); (left; hip)); ½; 0)))))))

2.  ((right; ankle);
     1;
     (1;
      (place; middle; forward);
      (right; hip);
      movement;
      ((whole body;
        (((right; hip); (left; hip));
         ((place; low; place); (place; high; place));
         (((right; hip); (left; hip)); ½; 0)))))))
```

All instructions are structured according to the descriptions given in Section 2.2.1. However, progression processor instructions are prefixed by two fields: one which designates the joint processor to which the instruction

will be dispatched, and one which gives the absolute time at which the instruction is to be executed. Initial placement modifiers are provided for all joint processors; the absence of further placement modifiers indicates that current description of the cross of axes is to be maintained. Also, all durations in this example are equal to one unit; in general, this will be a larger number, depending upon the temporal resolution required for describing other movements.

Movements of the two arms are easily achieved by displacements of the wrists since there are no other instructions which affect any other joints in either arm. These (joint processor, instruction) pairs therefore receive a low priority and are executed after support movements in each cycle. The two instructions to the progression processor each describe a forward "movement"; the center of gravity is moved forward from its present location by a fixed amount depending on the step length. Because a support movement requires a "preparation phase" [12], execution of the current instruction for a supporting joint depends on the instruction which follows. The progression processor must always look beyond those instructions it is currently executing to establish the proper context. In this example, the left heel strike actually occurs at the start of the segment; therefore, the position at this time will appear to be in the midst of the walk. When the final instructions to the progression processor are interpreted, it notes that there are no further instructions for the left ankle and therefore leaves the body balanced by bringing the center of gravity over the contact area of the right foot.

To achieve the appropriate leg movements implied by the support instructions, the progression processor generates contact instructions which are dispatched to the monitor. These contacts define the timing of the foot

movements and, together with the step length, the movement of the center of gravity, and the geometry of the supporting surface, implicitly define the joint angles at the ankles, knees, and hips. For example, to prepare for the step onto the right foot, the progression processor issues two contact instructions to the monitor: one to break the right foot contact with the floor at time 0.5, and the second to achieve a right heel contact with the floor at time 1.0 (the beginning of the first progression processor instruction to the right foot).

In this example a complex movement has been specified by a few instructions, but much of this complexity is accounted for by default conditions which may be overridden by additional detail in the instructions. By choosing a very small simulation interval, smooth animations can be produced. While this would divide each movement into many intermediate positions, there would be no additional overhead in the number of instructions actually sent to each processor.

3. THE BODY DATA BASE

We shall limit our discussion to movements which are skeletal, that is, which can be realized by a body model composed of joints and segments. Figure 3.1 is a more detailed version of Figure 2.1 with an itemization of all joint processors. The segments are drawn as lines connecting joint nodes. They are used in the data structure primarily to define coordinate systems and surface features of the body.

It will be useful to define a standard position for the body in order to fix certain coordinate system relationships which we shall use later. In the standard position the body is standing upright, all segments untwisted; feet flat, toes forward; and hands at sides, palms toward thighs (Figure 3.2). In our first encounter with the data base, we shall also find the body standing on the floor of a room, at the origin of a rectangular coordinate system. In this position the Z-axis of the room points "up" (that is, opposed to gravity), the X-axis points "forward" (as indicated by the arrow in Figure 3.2), and the Y-axis points "left." We shall frequently refer to room coordinates as "global" or "absolute."

The body is positioned with respect to the room through a chain of special instances of joints and segments. The room is regarded as such a segment establishing the global reference system. It only articulates with a single joint named "center of gravity." This joint, in turn, connects the room segment to a segment named "whole body." Finally, this "whole body" segment is connected to the root of the body tree, generally the center hip joint. If necessary, however, any other joint may be substituted as the root, since the graph structure of Figure 3.1 is undirected. (Figure 3.3 illustrates these relationships.)

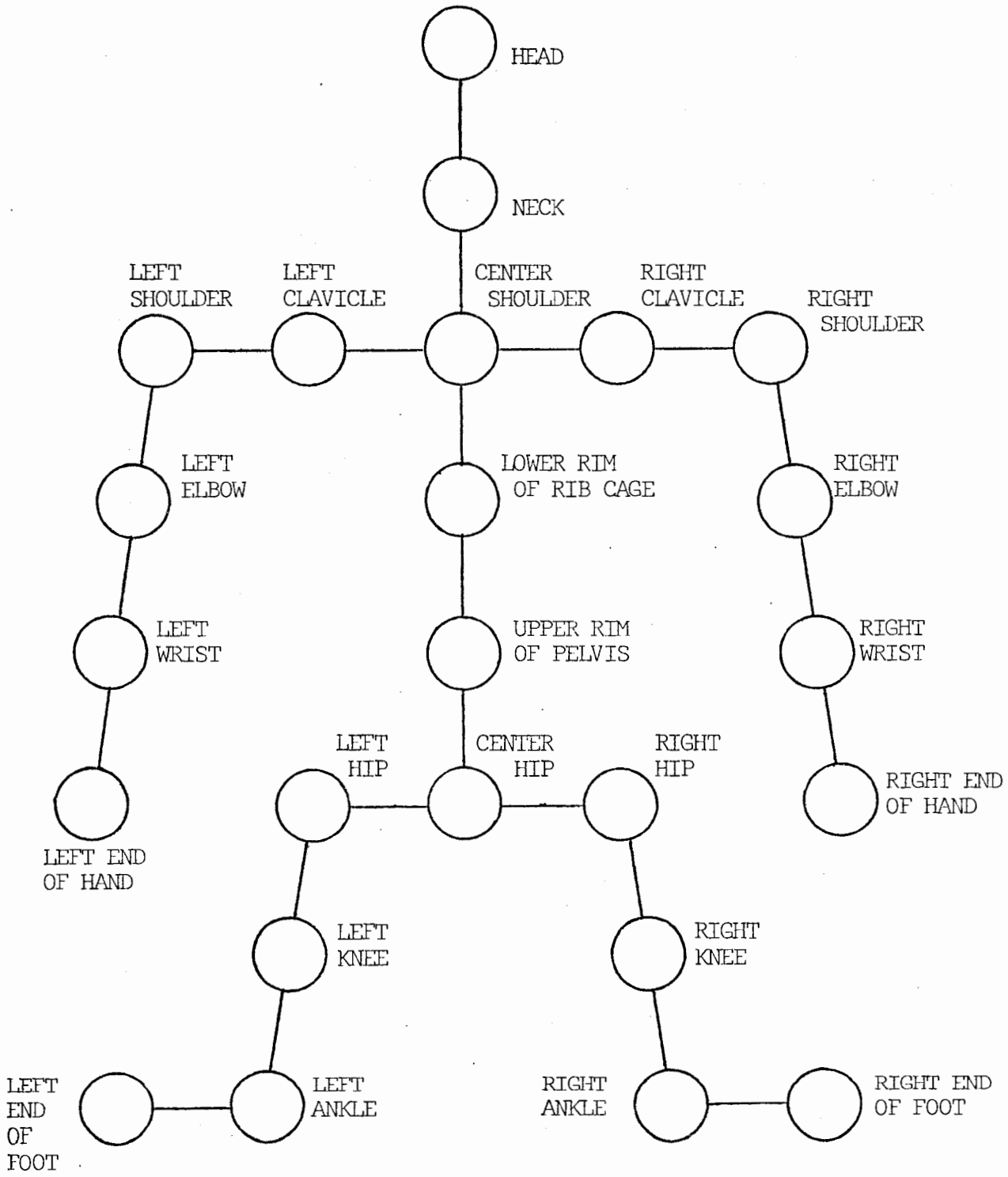


Figure 3.1 Joint processors and body tree.

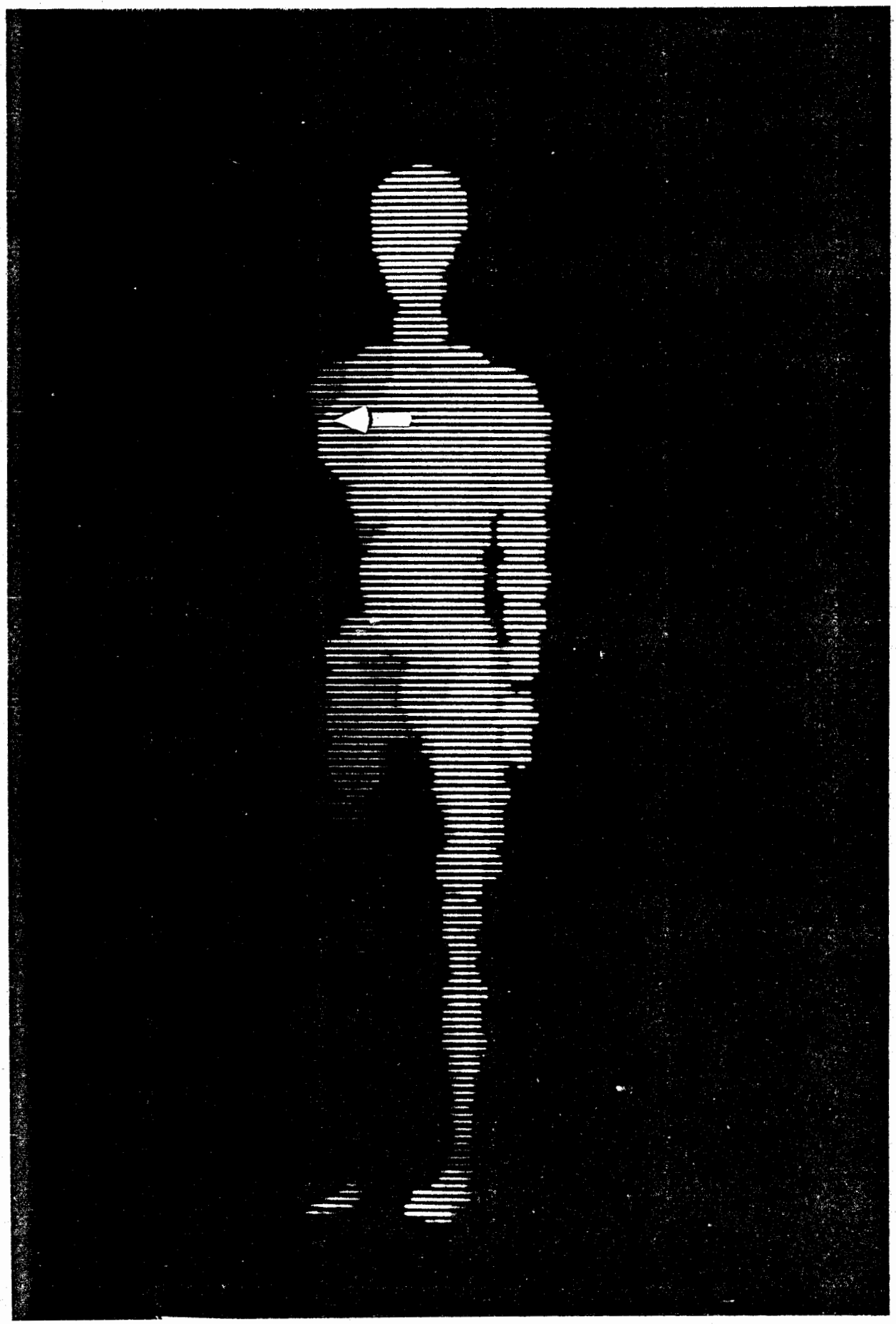


Figure 3.2 Standard position of body.

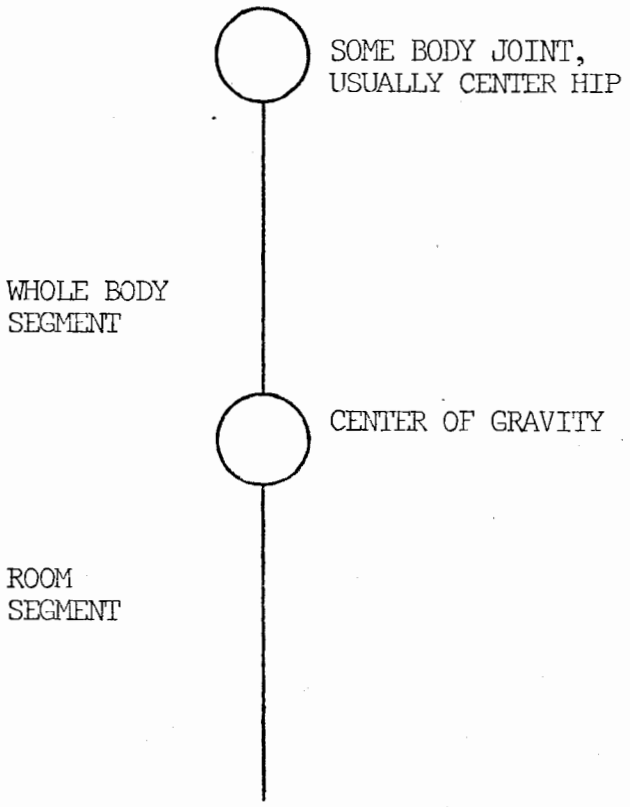


Figure 3.3 Relating the body to the room.

The center of gravity is represented as a "joint" and is maintained directly by the progression processor. It is the only joint allowed to translate with respect to its adjacent segment. The whole body segment permits movements of the body with respect to a line of direction; it effectively models the "stance" orientation described by Hutchinson [12].

3.1. Segments

Segments provide the rigid but articulated skeletal framework of the body and determine its external (surface) appearance. These functions are incorporated into the "segment" data type, whose components describe the relationship between the joints connected by a segment and also store information related to physical description of the associated body surface:

```
type segment = (name: sequence character;
                proximal: joint;
                distal: joint;
                distance: real;
                local directions: (forward: vector;
                                   left: vector;
                                   up: vector);
                twist: (positive limit: real;
                        negative limit: real;
                        current: real);
                orientation: vector;
                stop: proc (vector) logical;
                enclosure: sphere;
                centroid: vector;
                mass: real;
                surface: sequence sphere).

type sphere = (origin: vector;
               radius: real;
               features: sequence feature).

type feature = (name: sequence character;
                direction: vector).
```

Each segment has an internal name, given as a character string. Of the two joints connected by the segment, the one which lies closer to the root of the body tree (as defined above) is called the proximal joint and the other is called the distal joint. (A joint likewise connects proximal and distal segments.)

A cross of axes is defined for a segment such that the proximal joint is the origin and the ray connecting the proximal and distal joints defines the Z-axis direction. The coordinates of the distal joint are therefore (0,0 distance) where "distance" is the fixed length between the joints. The X-axis is chosen to lie in a vertical plane perpendicular to the global Y-axis through the proximal joint and also to have a positive forward component when the body is in the standard position (Figure 3.4). (If the Z-axis and the front direction coincide, then the X-axis is chosen to point upwards.) The Y-axis is the direction which yields a right-handed coordinate system.

As a segment moves, its cross of axes moves rigidly with it. Certain directions are defined in this local coordinate system to correspond to the conventional front, left, and up directions of the segment. For example, the negative Z-axis is up for the lower arm and the positive X-axis is front for the head, independent of the current orientation of these segments in the overall body position. Since a body segment may be capable of twisting along its Z-axis, the cross of axes may be rotated at the distal joint (Figure 3.5). Positive and negative rotation limits from the normal position of the X-axis (established above) define admissible twists.

A segment's orientation is given as a vector which is the position of the distal joint in the cross of axes of the proximal segment of the proximal

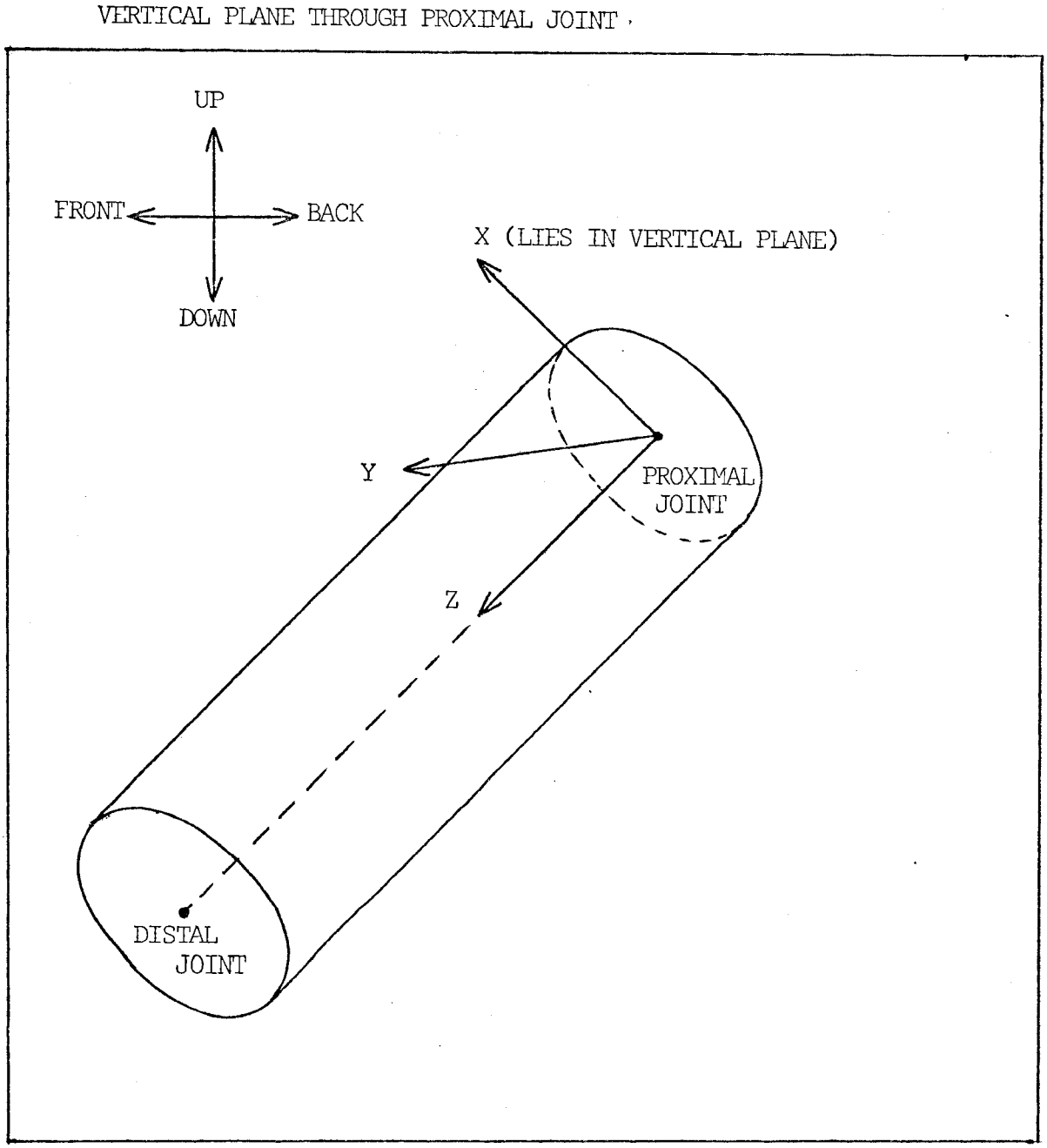


Figure 3.4 Segment cross of axes

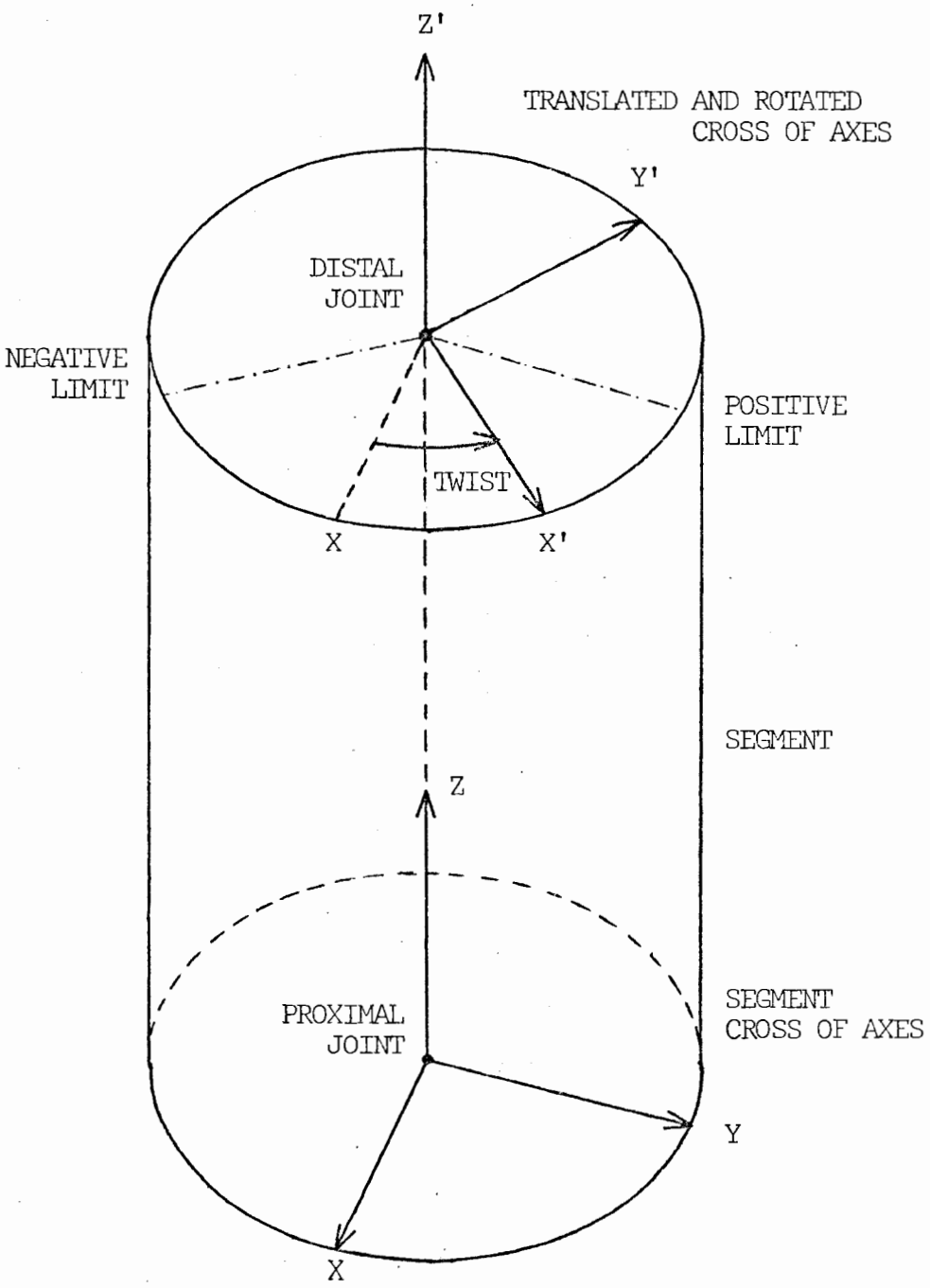


Figure 3.5 Affect of segment twist on cross of axes.

joint (Figure 3.6). Movements of the proximal segment are naturally transmitted through the common joint with no further computation. In addition, a "stop" function specifies whether or not a particular orientation is admissible. It may be used, for example, to limit movement to part of one plane, as at the elbow or knee.

Segments carry information to aid collision detection and support computations. The "enclosure" is the minimum sphere which includes the entire surface of the segment. It is used to approximate the location of the segment when comparing it against other (non-adjacent) segments and is fixed when the body model is dimensioned. A segment also has a fixed centroid (a vector in the local cross of axes) and a mass value which the monitor uses to compute the overall center of gravity for the body.

The "skin" (surface) of a segment is defined by a set of overlapping spheres [2]. The origin of each sphere is given as a vector in the segment's cross of axes. If the segment is twisted the sphere center is rotated about the Z-axis by an amount proportional to its distance along the segment from the proximal joint, which is just the Z component of the sphere's "origin" vector (Figure 3.7). Certain "features" of a segment may be distinguished by giving a point on a specific sphere a name which will allow it to be specified for a contact location or used in collision reports from the monitor. The feature direction is indicated by a vector in the segment cross of axes; normally this direction from the sphere origin will define the surface perpendicular (Figure 3.8).

3.2. Joints

Joints may be regarded from two points of view. First of all, they are distinguished points within the body which trace paths in space and determine

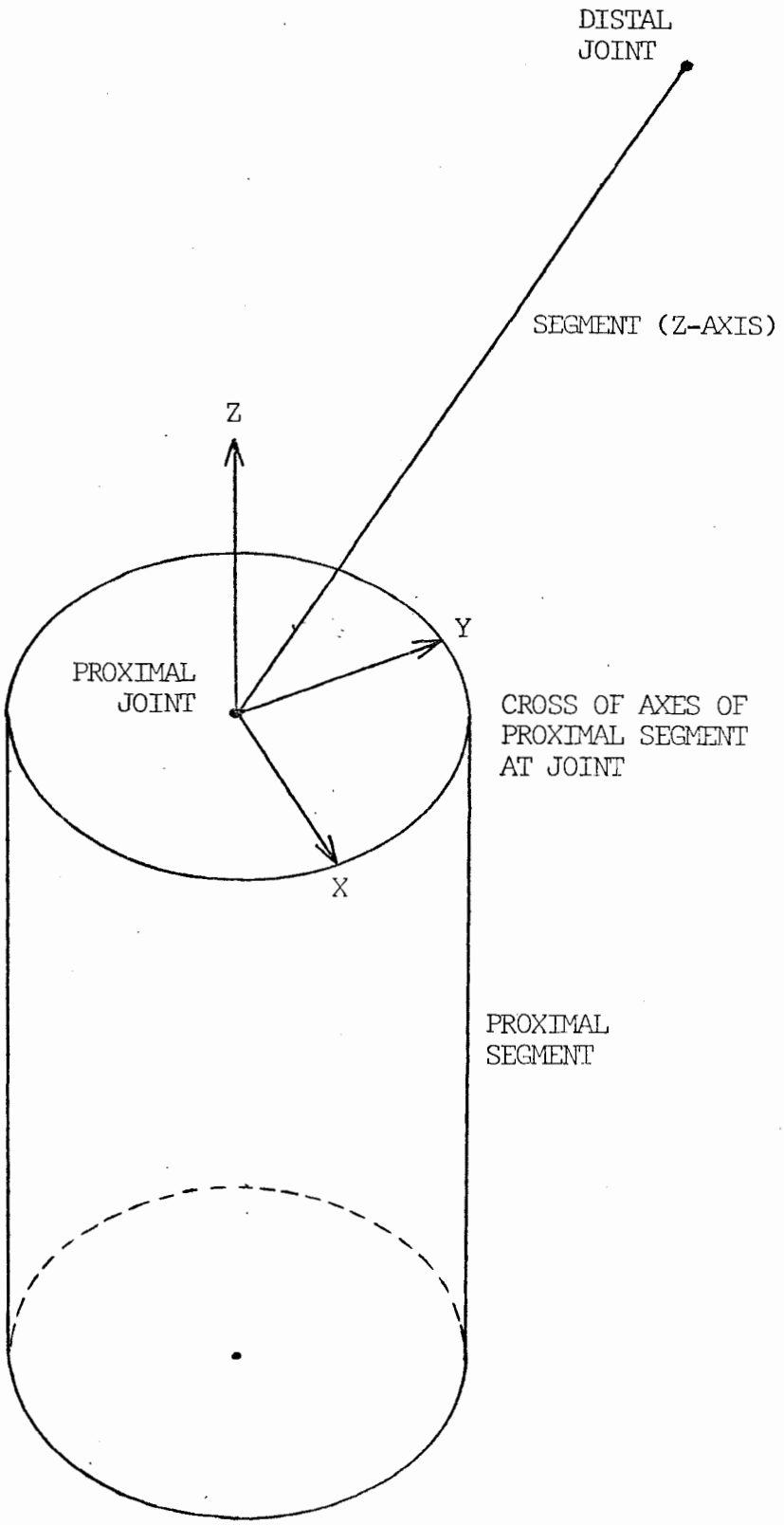
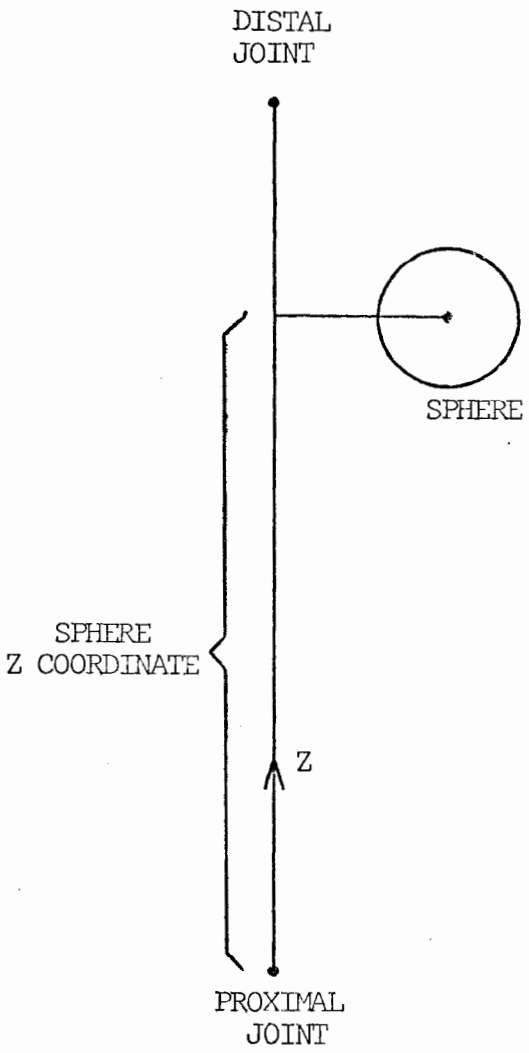


Figure 3.6 Orientation of segment with respect to distal segment.



$$\text{TWIST OF SPHERE} = \frac{\text{CURRENT TWIST}}{\text{Z OF SPHERE ORIGIN}}$$

Figure 3.7 Transmitting segment twist to the surface.

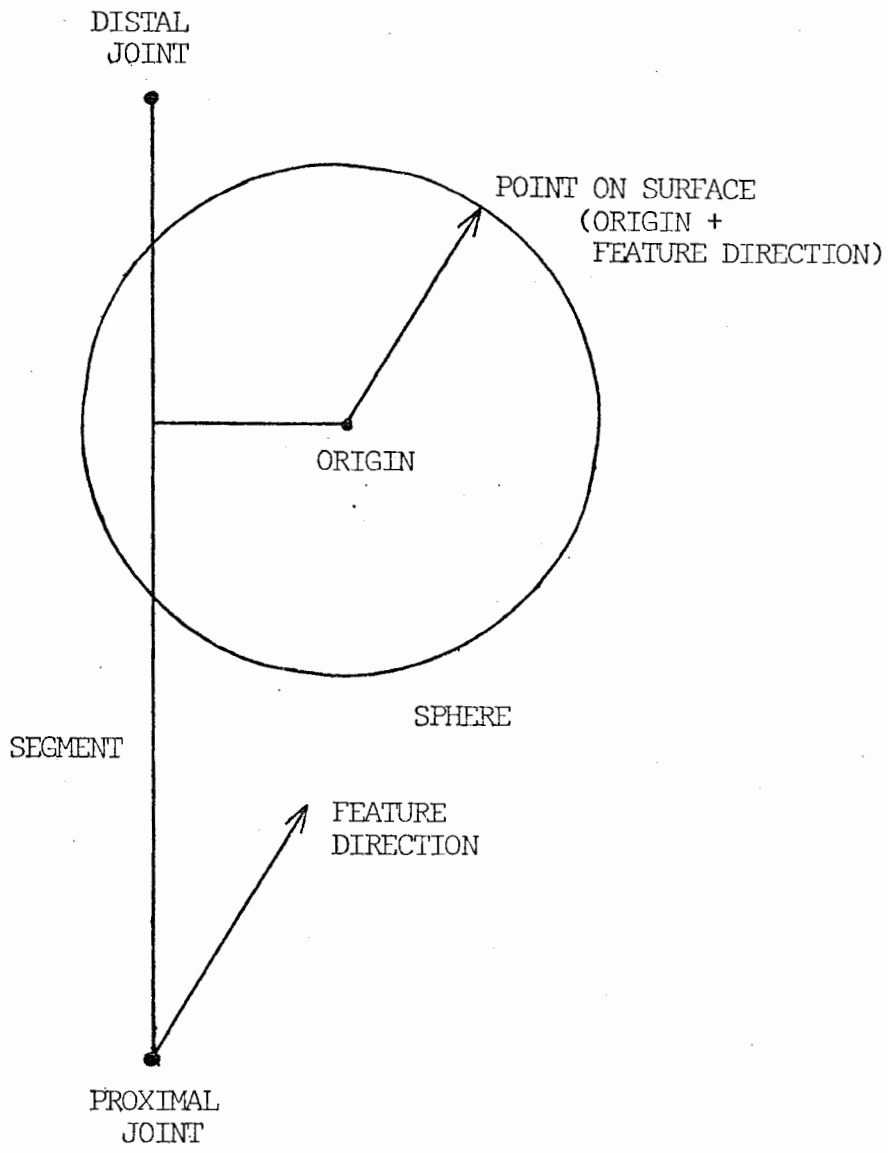


Figure 3.8 Specifying features.

how the entire body is positioned. Secondly, they are points of articulation of segments. At each joint the articulated segment which lies along the path connecting that joint to the root of the body tree (as defined above) is called the proximal segment of the joint. All other segments articulating at that joint are called distal segments. These two views of a joint are both incorporated in the following data structure:

```
type joint = (name:  sequence character;
              connection:  (proximal:  segment;
                            distal:  powerset segment);
              location:  (previous:  vector;
                          new:  vector;
                          forbidden:  sequence vector;
                          valid:  logical)).
```

The joint must connect a proximal segment with a non-empty set of distal segments unless it is an extremity. Finally, a joint has a character string name which serves as a label.

In Section 2.2.1 we defined position and movement descriptions for direction signs. In the simulator, these concepts are generalized to apply to any instruction. A position description indicates the location of the joint irrespective of any previous location (direction signs and shape descriptions of the "position" kind, facing signs, contact signs, and some revolution signs). A movement description indicates the joint location relative to its location at the start of the instruction (direction signs and shape descriptions of the "movement" kind and some revolution signs). Since an instruction is discarded by the joint processor after its interpretation is complete, it is necessary for the monitor to save the current locations of each joint for possible use in the next simulation cycle. At the beginning of each cycle, every valid "new" joint location is copied into the corresponding "previous" register. (Initially the

standard position of the body provides this data.) During instruction execution this previous location may be used to establish a joint location with respect to a particular cross of axes. This cross of axes may be different from that originally used to achieve the position. For example, there are different "holds" to preserve various aspects of a position [12]; and it is primarily for movement instructions which arise from "hold" conditions (Section 4.2.2) that the previous location is necessary.

For a movement description the "previous" location must be used to establish the desired goal location; although this information is obtained directly from a position description. In either case after the new position is computed, the "new" register receives the location of the joint in the global coordinate system and the "valid" flag is set to "true." The "valid" flags of all other joints within the scope of the instruction are set to "false" since their absolute locations may no longer be accurate. They will be brought up to date when that information is needed.

Once a joint is positioned, the "forbidden" register prevents further movement in one or more directions during the remainder of the simulation cycle. This register is initially nil (except for ankle and end of foot processors in standard position which cannot move downward any further). It is used to maintain contacts with parts of the room (such as the floor), to limit the movement of the center of gravity (so gestural movements will not perturb its path), and to execute multiply-constrained movements (for example, whole body twists while supported on both feet).

3.3. Data Base Management

Joint processors may obtain any information stored in the body data base upon request to the monitor. Only a limited set of modifications to the

data base are permitted, however, and these are also mediated by the monitor. The primary reason is that all movements are ultimately executed by general algorithms which are shared by all processors (via the monitor) [5]. These general algorithms are the data base "primitives" which the other processes use to implement their movement instructions. A second reason is that during the execution of these algorithms, certain body limitations may be reached, such as joint stops or segment collisions, and the monitor may be able to invoke some general strategy to finish executing the movement (such as twisting the proximal segment or trying an alternative movement path in a linkage situation). If these heuristics fail, control may be returned to the processor issuing the request.

The permitted movements are:

1. MOVE a joint to a point in a reference system from a given fixed end.
2. ROTATE a joint by a given angle about some axis from a given fixed end.
3. TWIST a joint by a given amount from a given fixed end.
4. BEND a joint to some angle.

Each of these functions is outlined below.

The MOVE function requires that one-, two-, or three-segment linkages be moved in space from an initial position to the given final position. One and two segment solutions are straightforward, while the three segment case involves choosing reasonable heuristics to select a solution among the many possible. All cases must take into account joint limits and forbidden directions. These algorithms and their implementation are discussed by O'Rourke [17]. TWIST, BEND, and ROTATE directly update the orientation and twist information on the distal segment of the indicated joint. TWIST

merely alters the twist angle, possibly twisting proximal segments if a limit is reached. BEND changes the orientation vector by changing the angle between the Z-axes of the segments adjacent to the joint. The ROTATE function specifies an axis of rotation about which the orientation vector is transformed. Other segments may be affected if the rotation attempts to move the joint into a region restricted by the stop function. In the case of either BEND or ROTATE, the twist of limbs which have significant freedom of rotation (at the shoulder or hip) are adjusted by a standard orientation function for the limb to conform to the conventions of Labanotation [12].

4. JOINT PROCESSORS

The joint processors are responsible for interpreting the active direction, revolution, facing, and shape description instructions during a simulation cycle. While the monitor provides some scheduling between the various processors, each joint processor is responsible for determining the proper sequence of actions needed to implement concurrent instructions with the same priorities (Section 2.3). We shall now describe the data structures maintained by each joint processor, after which we shall show how each movement or concurrency of movements is achieved.

4.1. Data Structures of a Joint Processor

There are three groups of registers within each joint processor. The first contains the instruction stream, the second consists of input sequence control, and the third is composed of register subsets specific to each instruction type. The first two sets will be described in this section, and the instruction registers will be described in the following sections.

As we briefly discussed in Section 2.3, instructions to a joint processor are formed into streams. Each stream consists of sequences of instructions or concurrent instruction streams. These are formatted so that only one level of parallelism is necessary:

```
type stream = sequence (substream, concurrency).

type concurrency = sequence substream.

type substream = sequence instruction.

type instruction = (direction signs,
                    revolution signs,
                    facing signs,
                    shape descriptions).
```

This structure is reflected in Figure 2.9. Within a substream all timing information is based on the sequence of instruction durations; likewise,

substreams and concurrencies are assumed to be sequential, without time "gaps."

Since each joint processor is responsible for handling an instruction stream, program control consists of one or more program counters. There is one program counter for each parallel substream of the instruction stream. Each program counter stores a delay value which indicates when the associated instruction substream must be "advanced," that is, the lead element deleted.

```
type program counter = sequence (delay:  rational;
                                 association:  substream).
```

Initially the joint processor contains a single program counter with a delay of zero.

At the beginning of a simulation cycle, the joint processor checks the program counters. For each program counter whose delay is zero, the following steps are repeated until the delay is nonzero:

1. Fetch and interpret the instruction in its associated substream. If the substream is nil, then there are two possibilities:

   1.1 If this is the end of a substream or concurrency, the old program counters are replaced by a new set, each referring to a new substream.

   1.2 If this is the end of the entire stream, the joint processor becomes dormant for the remainder of the simulation.

2. Increment the delay value by the duration (possibly zero) of the instruction just interpreted.

3. Delete the interpreted instruction from the substream. An empty substream is represented by nil.

An interpreted instruction is placed in the appropriate registers for its instruction type. After these registers are set, the monitor interrogates each active joint processor to determine the "scope" of each (joint processor, instruction) pair (Section 5.1) and the shortest (nonzero) delay among all the program counters. The simulation interval is the smaller of this delay value and a fixed minimum interval. Every program counter delay is then

decremented by the simulation interval. The monitor can now proceed to schedule instruction execution.

4.1.1. Direction registers

Interpretation of a direction sign depends essentially on the extraction of three pieces of information. Most important is the specification of a movement destination. This destination is interpreted with respect to an environment, which is constructed from the fields of the direction sign. Finally, it is necessary to keep track of the duration remaining for the achievement of the destination. Additional modifiers may be supplied to further define the movement. A set of direction registers thus has the following structure:

```
type direction registers = (goal:  destination;
                            env:   environment;
                            duration remaining:  rational;
                            modifiers:  powerset constraints).
```

The environment has the following structure:

```
type environment = (cross:  reference;
                    fixed end:  joint;
                    augmented scope joint:  joint;
                    current position:  vector).
```

The "fixed end" register is filled directly from the "proximal joint" field of the instruction; and "cross" is established by a placement modifier or, if a placement modifier is not specified, by a default associated with the joint processor. The "augmented scope joint" is normally the same as the fixed end but may be altered by the joint processor in response to specific commands (inclusions in a direction sign) or specific movements (destinations which physically force inclusions). It is used by the monitor to establish an instruction's priority (Section 5.1).

The "current position" is obtained from the monitor as the location of

the joint relative to the "cross". The monitor must use the latest information in establishing the reference; hence the "current position" will take into account the results of any other movements which have already been executed during this cycle. This may force the computation of the absolute locations of any required joints which are currently flagged as not valid.

The system of reference for an environment is given by the following substructure:

```
type reference = (origin:  (joint, vector, local vector);
                  orientation:  (forward:  direction;
                                 left:  direction;
                                 up:  direction)).

type local vector = (segment;  vector).

type direction = (fixed:  (vector, local vector),
                  body:  (from:  (joint, local vector);
                          to:  (joint, local vector)),
                  tangent:  (real; real)).
```

A system of reference is established by locating its origin and giving the directions of its three coordinate axes. The location of the origin may be given in terms of a joint, a vector of absolute coordinates, or a vector defined in terms of the local cross of axes associated with a segment. Similarly, axis direction may be given by a vector, either in absolute coordinates or in terms of some segment's local cross of axes; alternatively, the direction may be given by a line connecting any two points of the body or by the tangent to the current path (given as a vector in the plane of the floor).

The structure of "destination" depends on whether the "kind" field of the direction sign is "position" or "movement":

```
type destination = (position:  (to:  vector;
                                path:  (straight, radial)),
                    movement:  (toward:  vector;
                                from:  vector)).
```

All vectors are interpreted with respect to "cross" as provided by the environment. If the destination is a "position", the joint will be moved "to" that position from its "current position" along a straight or curved ("radial") path. A "movement" description is always interpreted as being in a straight line from the location of the joint at the start of the instruction interpretation. (This location, expressed in terms of "cross", is stored in the "from" register.) The magnitude of the movement vector ("toward") is determined by either the joint processor (for gestural movements) or the progression processor (for support movements).

Finally, we have several possible modifiers:

```
type constraints = (contacts, deviation, bends).

type contacts = sequence contact block.

type deviation = proc (real) vector.

type bends = sequence (location:  joint;
                       surface:  local vector;
                       begin:  rational;
                       duration:  rational;
                       achieve:  (fraction, nil)).
```

Information regarding contacts is supplied by the monitor as part of the interpretation of contact signs. We shall pursue this further in Section 5.2. The directional displacement is a "deviation description" (Section 2.2.1) represented as a vector-valued function of time, where the vector output is defined in terms of the system of reference given by the environment. We shall see in Section 4.2 how this function is integrated into the actual movement implementation.

Bends account for the remainder of the information specified by a direction sign. The affected joints will normally include all body joints between the augmented scope joint and the joint processor. Omitted joints

are assumed to be unaltered by this particular instruction. Each joint is given a bending surface, a starting time, and the length of time allowed until the bending angle is achieved. The bending surface is a local vector which translates into a rotation axis at the joint, and the angle is expressed as a fraction of the total possible bend in that direction; this value can be computed by the monitor from the current joint position and the joint stop function. A value of nil in this register is interpreted as "whatever bending is necessary" and can be used to implement multiple bends whenever explicit bends have not been specified [17]. By including joints in the bending list which lie further from the joint than the fixed end, inclusions (Section 2.2.1) may be implemented. The degree of inclusion is translated into default bending fractions which will be used as maxima, not necessarily as values to be strictly achieved.

The bending registers are also used to effect most "intermediate joints" modifiers (Section 2.2.1). Leading and following are implemented by small bending movements followed by bends in the opposite sense. Inward and outward successions are sequences of such bends and cancellations. Default values for these movements are established by the processor whenever they are not specified in the instruction. Guidance by a body surface is handled differently: a facing sign is generated by the joint processor to orient the surface toward the direction of travel during an initial fraction of the direction sign.

4.1.2. Revolution registers

Revolution signs require for their interpretation the same timing and environment information as do direction signs. In this case, however, the movement is represented as the remaining orientation change about an

axis, along with a modifier indicating whether this change is to be achieved as a twist or a rotation:

```
type revolution registers = (env: environment;
                             duration remaining: rational;
                             axis: vector;
                             change remaining: real;
                             revolution modifier: (twist, rotate)).
```

Orientation changes are specified as multiples of a full revolution about the "axis", expressed as a vector in terms of the environment "cross" Initially the "change remaining" value is the same as the "amount" field of the revolution sign. The "sign" value is subsumed into the "axis" vector direction so that the rotation or twist follows the right-hand rule. We shall make the restriction that the fixed end of the revolution be identical to the origin of the environment "cross", except for instructions to the progression processor. Also, for a twist the joint itself must lie on the axis.

Revolution may describe particular positions or relative changes in the orientation of a joint with respect to its current "fixed end". These cases are distinguished by the "origin" field of the revolution sign. If this value is "stance" or "absolute", an appropriate direction is computed from the "sign" and "amount" values. This direction is then used as the "direction" field in a facing sign generated by the joint processor. The facing sign replaces the revolution sign and inherits its other attributes, transformed as necessary.

When the revolution sign "origin" is "current", a relative angle change is assumed. The joint processor will execute the total amount of change requested, even if other instructions augment or diminish the apparent movement. For example, the orientation of the proximal segment may change.

While this may induce an additional twist, Labanotation conventions [12] preclude any alteration to the "change remaining" register.

4.1.3, Facing registers

The interpretation of facing signs requires nearly the same information as is provided by revolution signs. This makes sense, since a facing sign invokes twists or rotations indirectly and is the position description analog to a revolution sign:

```
type facing registers = (env:  environment;
                         duration remaining:  rational;
                         facing modifier:  (twist, rotate);
                         action:  facing).

type facing = (area:  vector;
               goal:  vector;
               axis:  vector).
```

The "goal" register stores the orientation which the surface "area" is to achieve. Both vectors (as well as the "axis") are expressed in the local coordinate system of the segment containing the surface to be oriented. The origin of this system will then be situated at the joint whose processor received the instruction.

The "axis" vector is derived rather than supplied by the instruction. It is chosen as that perpendicular to the plane formed by the "area" and "goal" vectors such that the former is rotated toward the latter through an acute angle. (If the other direction is desired, a simultaneous revolution or an intermediate facing must be used.) If the two vectors are directly opposed, then the axis is chosen to lie as close as possible to the Z-axis of the distal segment containing the surface area. If this cannot be done (the vectors are parallel to the Z-axis), the X-axis of the segment is chosen arbitrarily. If the two vectors represent the same direction, then the joint has achieved the facing goal. Since a fixed end is not provided in

the facing sign, the joint processor itself is used. The facing is then implemented as a rotation or twist of the distal joint of the segment containing the surface about the "axis". Facings of the "twist" type can only arise from modified revolution instructions (Section 4.1.2).

4.1.4. Shape registers

Shape descriptions, like direction signs, may be either position descriptions or movement descriptions:

```
type shape registers = (env:  environment;
                        duration remaining:  rational;
                        action:  shape).

type shape = (kind = (position, movement);
              configuration:  proc (real) vector).
```

For a position description the body parts between the joint processor and its fixed end are to achieve some configuration in space. This is given as a vector function whose input is the distance along the path connecting the processor's associated joint to the fixed end. The configuration may then be achieved by iteratively fitting body segments to the shape. A movement description, on the other hand, describes the path of the joint associated with the processor for the indicated duration. The "configuration" is now interpreted as a vector function of time, appropriately scaled so that the final position is achieved at the end of the instruction.

4.2. Increment and Destination Computation

The most elementary function of each joint processor is to move its joint towards a destination point, direction, or orientation during each simulation cycle. At the beginning of a simulation cycle, the joint determines its current position with respect to the appropriate reference system. This is necessary because the joint may have moved from its goal at the end of the previous cycle; other processors or changes in the location or orientation of the reference system could have moved it. We shall assume

that the destination (or final goal) of the movement is known. How much the joint will move depends on the simulation interval;

```
if duration remaining ≤ simulation interval
   then do "move" from current position to destination;
           delete instruction; (it's completed)
           end
   else do "move" (simulation interval/duration remaining)
           of the way from current position to
           destination;
           duration remaining = duration remaining
                              - simulation interval;
           end
```

The definition of "move" varies from instruction to instruction. It will be helpful to define three functions to perform various interpolations, NVRI, NVLI, and NPRI:

```
NVRI: proc (current: vector; destination: vector; amount: fraction)
                                    returns (vector).
```

This procedure (New Vector; Radial Interpolation) returns the vector which is the fractional amount of the positive rotation from the current vector to the destination vector. The length of the result is linearly interpolated between the current and destination vector lengths (Figure 4.1).

```
NVLI: proc (current: vector; destination: vector;
            amount: fraction) returns (vector).
```

This procedure (New Vector; Linear Interpolation) returns a vector which represents the fractional amount of distance from the current to the destination vector (Figure 4.2).

```
NPRI: proc (current: vector; axis: vector; total: real;
            amount: fraction) returns (vector).
```

In NVRI the axis of rotation is implicitly perpendicular to the two vectors. For NPRI (New Position; Radial Interpolation) the axis is provided. The result is the current vector rotated about the axis by the fractional amount of the total rotation desired (Figure 4.3).

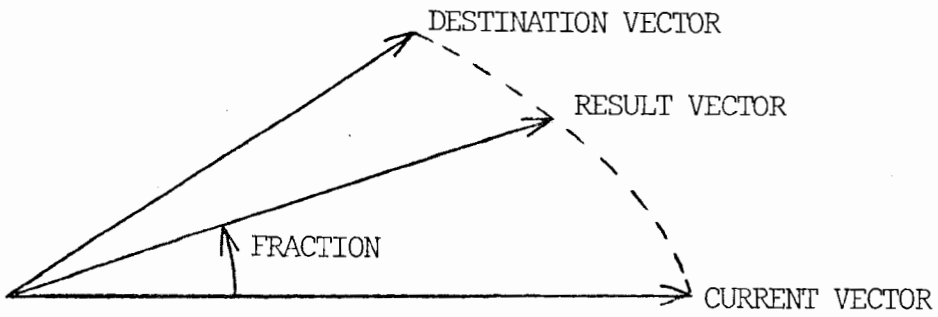


Figure 4.1 Interpolation with NVRI.

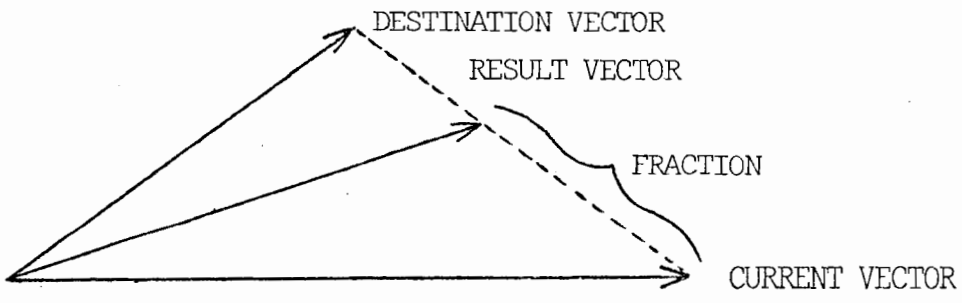


Figure 4.2 Interpolation with NVLI.

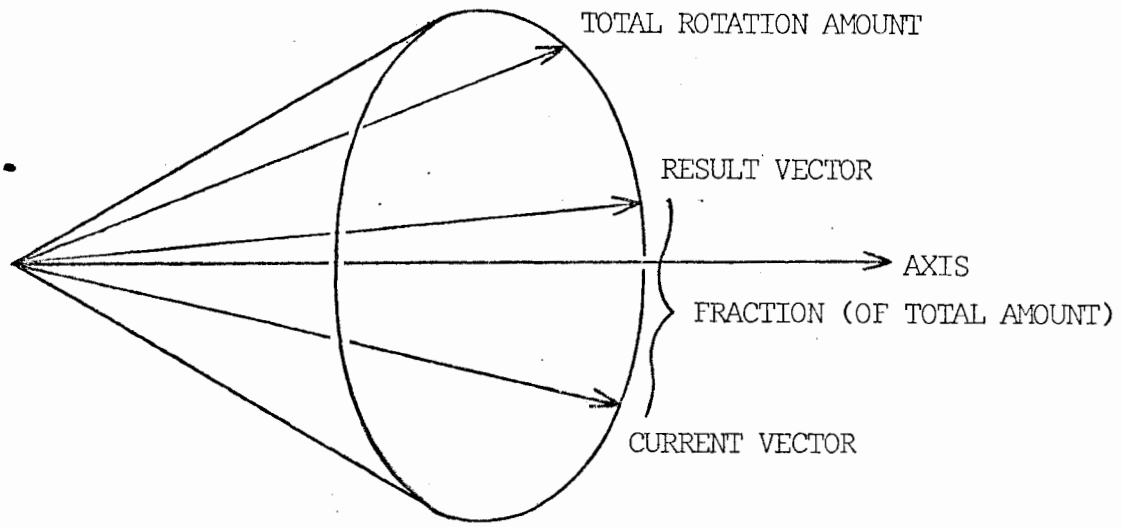


Figure 4.3 Interpolation with NPRI.

Although these functions linearly interpolate to compute the result vector, we can scale the fraction by any function F such that F(0)=0, F(1)=1, and $0 \leq F(x) \leq 1$ for $0 \leq x \leq 1$. The identity function F(x)=x is implicit in the above; but other non-linear functions may be used to emphasize smooth accelerations and decelerations [7,16,22]. We are avoiding this route for the present, however, since phrasing information is not specifically encoded into the instructions to the simulator.

4.2.1. Position computation

For a position description direction instruction we must determine which type of path is to be used. The following decision procedure has been abstracted from Labanotation [12]:

Δ = arc length distance between current position and goal;

if $\Delta = 0^{\circ}$
    then path = straight;
        use NVLI for interpolation;
if $0^{\circ} < \Delta \leq 90^{\circ}$
    then path = radial;
    use NVRI for interpolation;
if $90^{\circ} \leq \Delta \leq 180^{\circ}$
    then path = straight;
        include a slight deviation
        (depending on joints and position);
        use NVLI, then compute deviation;

Changes of the first type arise when position and goal lie along the same direction or when either is zero in the current reference system. The direction change in the second case does not, in itself, cause a change in the distance between the joint and its fixed end. In the third case, however, length changes are mandatory and must be compensated by bending intermediate joints. The timing and degree of bending are stored in the bend registers of the direction instruction. Both this information and the indicated deviation are supplied by the joint processor depending on its physical

capabilities and the current and goal directions.

For shape descriptions of the position type, the movement process is one of matching joint angles to the path of direction vectors. Starting at the fixed end, successive joints are adjusted by radial interpolation (NVRI) to approach the tangent to the shape at the appropriate distance. (The fixed end is "pinned" to the start of the shape; thus the first adjacent distal joint is the first to be moved.) The arc length of the shape can be compared to the total body length between the joint processor and the fixed end (obtained from the monitor), so that the distances involved are easily scaled.

Revolution and facing instructions interpolate rotations with NPRI. Additional effort must be expended, however, in insuring that intermediate joints rotate, or segments twist, by the appropriate amounts. For the elbows, wrists, knees, and ankles we can use the "universal joint" property of these hinge connections to transmit a rotation in a distal segment to a twist or rotation of equal magnitude in the proximal segment. This is not true for joints in the torso because of the pivot-like spinal connection; nor is it true for the ball joints of the hips, shoulder and head. Generally, these joints "absorb" rotations up to their physical maximums.

An additional problem must be confronted in twists or facings which include more than one fixed support or contact point, for example, a twist of the center hip with respect to the (fixed) foot positions on the floor. We cannot simply distribute these twists over the leg segments. Instead, a line connecting the left and right hips is rotated; and the positions of the knees are computed based on the hip and ankle locations. These positions

are then used to compute the actual relative orientations at the knees and hips and the twists of the lower legs. If one or both feet are not restricted to immobile support contacts then some other action any occur. For example, if one foot is not supporting body weight then the normal twist calculations can be performed on the supporting foot. These and similar situations are processed by the monitor since they are caused by various external constraints on joint positions, namely contacts.

4.2.2. Movement computation

Movement descriptions have two possible interpretations, depending on whether or not a direction of movement is specified. When it is given, the joint moves in that direction along a straight path. Intermediate positions are interpolated using NVLI with a "current" location of (0,0,0). Although the direction of movement is given in the direction sign, the length of the "destination" vector must be determined indirectly. Most direction instructions specifying movement alter support and are therefore handled by the progression processor. The "destination" vector is determined by the default step length or an appropriate modification of the default by a bend or stretch indication in the instruction. For non-support movements the magnitude of the movement may depend on a maximum displacement or some reasonable default movement rate.

When no direction is specified the instruction is interpreted as a "hold"; that is, the joint is expected to maintain some fixed relationship to a given reference cross of axes throughout the duration of the instruction. In this case, the position must be re-established every simulation cycle rather than achieved only at the end of the duration interval; so the location of the joint is used as the "current" vector and also as the "destination"

vector in NVLI. Any fraction of the total duration will therefore achieve the same position.

For shape descriptions movement is specified by a three-dimensional path of points. As we have previously noted, we will assume that the intent is to distribute the movement evenly over the total length of the path. It is therefore easy to compute the total path length over the point set, if only as the summed linear distances between the points. (Cubic interpolations could be used to "smooth" the path, making the path length that of the interpolated curve.) At any simulation time the direction of movement is toward that point on the curve lying (length/simulation interval) distance away from the current point along the path. For long simulation intervals the expected shape may be somewhat distorted; but as these are again primarily paths for the progression processor, we can expect the simulation interval to be short with respect to the total time needed to traverse the path.

4.3. Simultaneous Instruction Execution

During a simulation cycle a joint processor may be executing more than one instruction affecting the same sequence of body parts. A reasonable execution sequence for these instructions must be determined, although conceptually they are to be executed in parallel. Among concurrent instructions the default execution order is:

1. shape descriptions of "movement" kind
2. shape descriptions of "position" kind
3. revolution signs of "rotation" kind
4. revolution signs of "twist" kind
5. facing signs

6. direction signs of "movement" kind
7. direction signs of "position" kind

The general rules used to construct this ordering can be summarized:

* Shape descriptions are executed first since they define global movements or configurations of body parts.
* Revolution signs are executed next since they tend to orient limb units.
* Facing signs are executed next since they may cause joint rotations or twists to achieve the facing direction.
* Direction signs are executed last since they move joints to specific locations in space, subject to the constraints established by the preceding instructions.

In addition, the following factors were influential:

* "Position" instructions are executed after "movement" instructions, since a position must be achieved regardless of the movements which co-occur with it.
* Rotations are performed before twists because rotations do not affect intermediate joints; facing signs and twists do, so they must be concerned with the admissible positions and movements of the intermediate joints.
* Direction signs must be executed after facing signs to assure that contact processing will establish the desired relationships (Section 5.2.2).
* "Position" direction signs are executed after "movement" direction signs so that contacts (maintained by "movement" direction signs) may be broken by subsequent positioning instructions.

The modifiers of a direction sign are applied to the basic direction movement. The deviation function is applied to obtain an absolute displacement which is added to the current position. From this adjusted position bends are executed, proceeding along successive joints toward the augmented scope joint. Finally, contacts are approached or maintained according to the process described in Section 5.2.

5. MONITOR

In Section 2.3 we indicated how the monitor structures the overall control flow during a simulation cycle. We have noted that the monitor manages the body data base and provides computational utilities for maintaining and modifying the data base in response to requests from the other processors. In this section we shall discuss the remaining duties of the monitor: scheduling the current set of concurrent processes, and moderating the achievement and maintenance of contacts.

5.1 Priorities of Processing

Since the monitor maintains the body data base, it must also be responsible for the order in which changes are allowed during a simulation cycle. Because the body must remain connected, all movements (with the exception of movement of the body as a whole) must be realized by rotations. This is achieved by changing the orientation vector of the individual segment data structures. As we have noted, however, rotating one joint may invalidate the absolute positions of all other joints which lie "beyond" that joint in the body tree (i.e. further from the root). Since a processor controlling one of these joints may be executing an instruction, it is imperative that that processor have current information on the location of its joint.

During a simulation cycle the monitor must schedule the active processes to insure the determinism of the resulting movement. Thus, the semantics of a particular set of concurrent instructions should not be different from one execution to the next. Put somewhat differently, the individual joint processors determine the semantics of the instructions they are currently executing, while the monitor determines the semantics of the collection of executing processes with respect to the structure

of the body.

It is therefore necessary to determine a priority for the order of execution of all concurrent instructions. This priority is based on whether one subtree of the body tree is contained in another, and which of these are the "largest" among the current set of trees. The scope of an instruction is defined as the subtree of the body tree formed by the directed path from the fixed end of the instruction through the joint receiving the instruction, followed by the remainder of the body tree rooted at that joint (excluding the path already defined to the joint). We shall call a particular kind of subtree an augmented scope tree if the initial path is rooted at the augmented scope joint. For example, if an instruction to the center shoulder specifies the center hip as fixed end, but the left hip is the augmented scope joint, then the augmented scope tree is shown in Figure 5.1.

The instruction priorities can now be derived from relationships between the set of augmented scope trees. We first define a maximal instruction and compute priorities among the set of maximal instructions. A maximal instruction is one whose augmented scope tree is completely contained in no other augmented scope. The remaining instructions can be assigned priorities through an ordering defined among the subtrees of an augmented scope tree (based on containment of one subtree in another). A maximal instruction augmented scope may still intersect that of another instruction, so we distinguish an isolated maximal instruction as one whose augmented scope is disjoint from or else shares only the root joint with any other maximal instruction. Otherwise the maximal instructions overlap: at least one edge (segment) of the body tree is common to both.

An isolated maximal instruction is independent of any other maximal

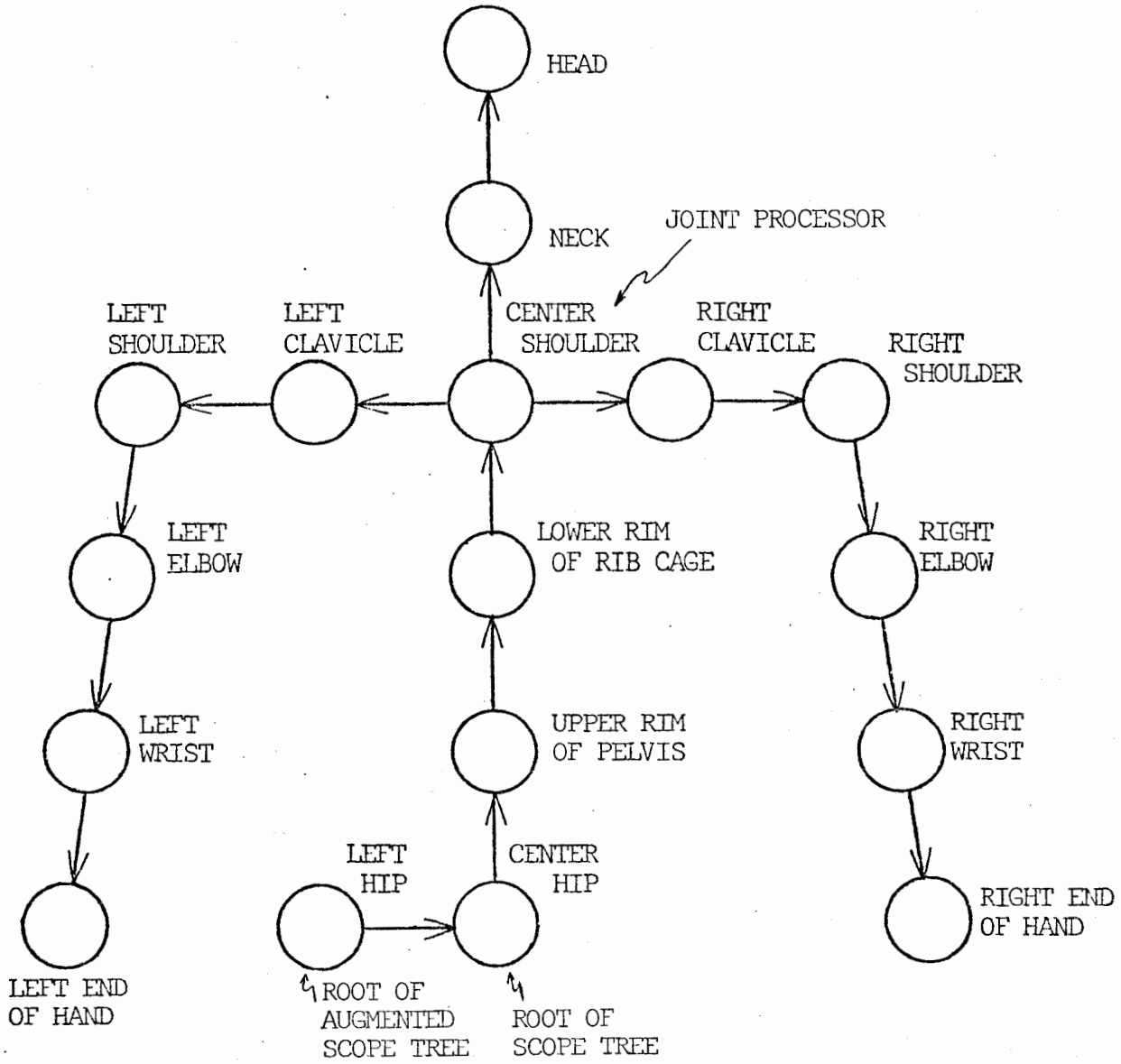


Figure 5.1 Example of augmented scope tree.

instruction and receives the lowest priority. To differentiate among overlapped maximal instructions, we use two heuristics:

1. Assign higher priority to the maximal instruction having a supporting joint within its augmented scope.
2. Assign higher priority to the maximal instruction having a passive contact point within its augmented scope (Section 5.2.1).

These are applied so that (1) has precedence over (2). The first heuristic arises from the fact that a supporting joint carries weight and cannot be moved with respect to the point of support; thus the remainder of the body must be positioned relative to that constraint. The second heuristic derives from the execution of contacts: an active contact point must adjust to the movements of a passive partner contact point and therefore must be moved last. In case of ties, a choice is made arbitrarily, although it is expected that this case will arise infrequently: overlapping scopes are apt to be ambiguous even to a human interpreter.

Once the priorities of the maximal instructions are established, priorities for the remaining instructions depend only on subtree containment. We will also adopt a "depth-first" ordering such that all instructions "less than" a maximal instruction of highest priority are executed before the maximal instruction of next highest priority. The non-maximal instruction priorities therefore fall "between" those of the maximal instructions. Consider two trees contained within the augmented scope of another instruction. They may intersect in one of seven ways (Figure 5.2):

1. Same joint processors and augmented scopes. The order of execution is determined by the joint processor, not the monitor (Section 4.3).

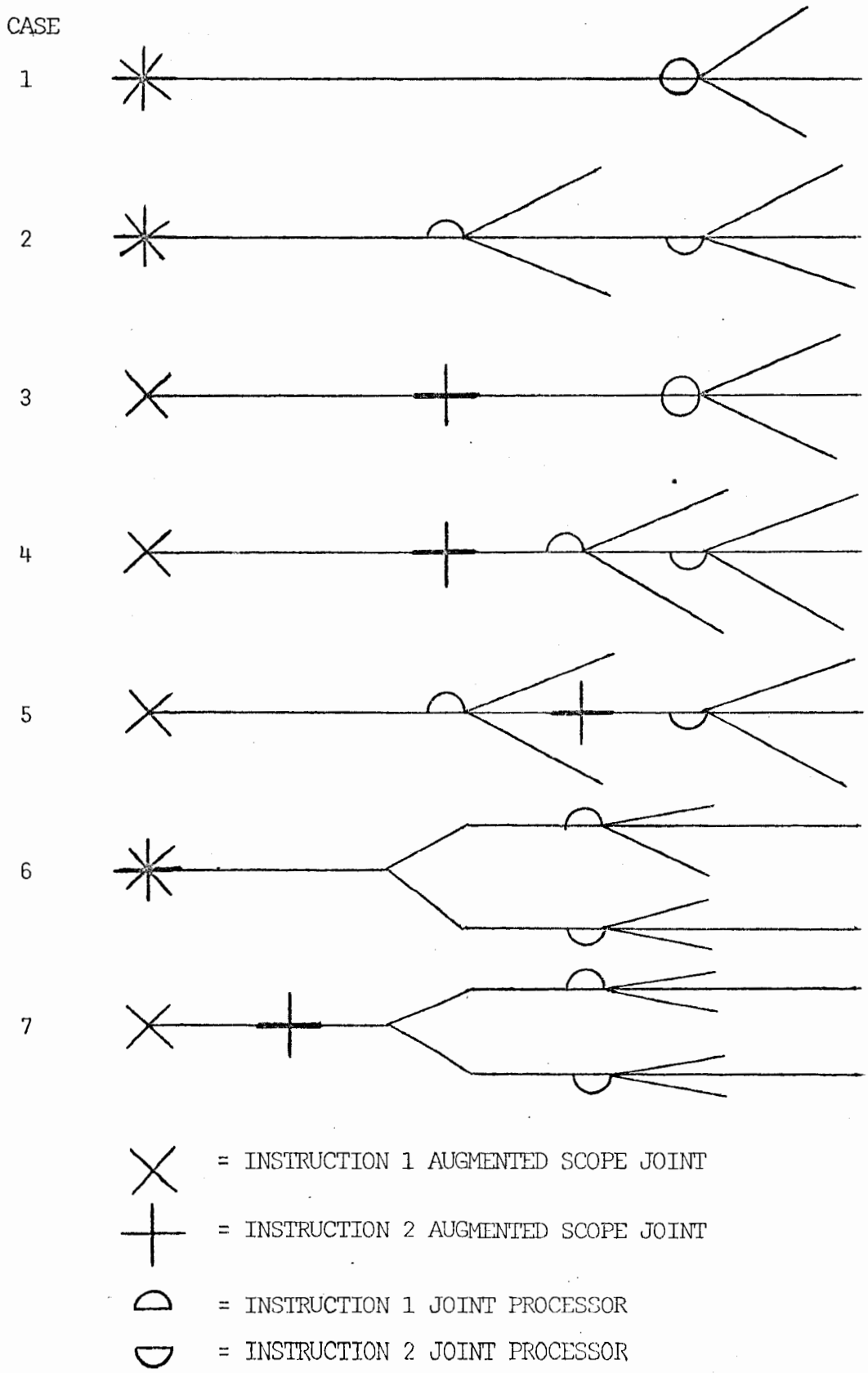


Figure 5.2 Augmented scope overlap cases.

2. Same augmented scope, but joint of instruction one lies on path between augmented scope joint and joint processor of instruction two. Instruction one is executed first.
3. Same joint processor, but augmented scope joint of instruction two lies between augmented scope joint and joint processor of instruction one. Instruction one is executed first.
4. Different augmented scope joints and the joint processor of instruction two lies in the subtree rooted at the joint processor of instruction one. Instruction one is executed first.
5. Different augmented scope joints and joint processors, and the augmented scope joint of instruction two lies in the subtree rooted at the joint processor of instruction one. Instruction one is executed first.
6. Same augmented scope joint, but joint processors are roots of disjoint subtrees. The relative priorities of the instructions are computed as if they were maximal instructions using the heuristics described above.
7. Same as case 6, but with the augmented scope joint of instruction two between the augmented scope joint and joint processor of instruction one. Instruction one is executed first.

5.2 Contact Processing

Contact instructions are handled by the monitor since contacts may occur between arbitrary body surfaces. We shall make the assumption that contacts are essentially local phenomena, in the sense that suitable instructions of the other types will approximately position the involved surfaces. We can therefore avoid defining a unique scope for a contact instruction; rather, its effective scope will depend upon the instructions currently being executed by the affected processors.

Contact signs in the monitor input stream are read and interpreted

at the very beginning of the simulation, since the monitor cannot know a priori what subsequent direction signs will be used to actually implement the contact. The contact signs are compiled into contact blocks, one for each pair of objects in the "contacts" sequence of the instruction:

type current contacts = sequence contact block.

Whatever information cannot be compiled is inserted when suitable direction signs are found which overlap the contact time. Many contact signs are generated during the simulation by the progression processor, and the monitor simply assimilates them into the "contacts" data structure as soon as they are received. Achieved and inactive contacts are deleted.

The goal of contact processing is to convert the relational specification in a contact sign into explicit points, distances, and directions. Achievement of a contact is thus dependent upon bringing two points together within a certain distance tolerance. Each contact block describes a single relationship between two points (not necessarily fixed) on the surface of the body or on an auxiliary object. For each active contact point (there must be at least one for each contact block), a joint processor is delegated responsibility for moving the point to achieve the correct contact. These joint processors have (or else will be supplied with) a suitable direction instruction whose "contacts" register (Section 4.1.1) will refer to the contact block. The "influence" registers are used to control the rate of contact achievement. The "duration interval" registers contain the difference between the contact time and the starting time of each direction sign with its contact modifier:

```
type contact block = (timing: (at: rational;
                               duration remaining: rational;
                               until: (rational, nil)));
                     relationship: (upper limit: real;
                                    lower limit: real;
                                    initial distance: real;
                                    current distance: real;
                                    difference  vector: vector;
                                    weight: (real, nil));
                     contact 1:  (place 1: virtual contact;
                                  delegate 1: joint;
                                  influence 1: real;
                                  duration interval 1: rational);
                     contact 2:  (place 2: virtual contact;
                                  delegate 2: joint;
                                  influence 2: real;
                                  duration interval 2: rational;
                                  role: (active, passive))).
```

Timing information consists of the time at which the contact occurs ("at"), the "duration remaining" until the contact time, and the time when the contact is to terminate. The "until" register may contain an explicit termination time, determined from a later contact sign (see Section 2.2.4), or nil, in which case movements of the joints will break the contact naturally. The "at" and "until" registers will hold the same value when the contact is "in passing".

Because a contact sign need not describe an actual touch relationship between two contact points, we must allow the specification of any point related to a body or object surface. The "places" of contact are "virtual" points described by the "virtual contact" data structure:

```
type virtual contact = (source: (body: (which: segment;
                                         using: (sphere, nil)),
                                 other: object);
                        location: (fixed, sliding);
                        direction: (vector, nil);
                        point: vector).
```

The contact point may be associated with a body segment or another object. In the former case, a specific sphere may be used; or the contact point may

"float" about the segment surface. The "location" of the contact point may be fixed or may be free to slide about within the prescribed domain. (For example, if a fingertip sphere is specified in a hand segment, then a sliding "location" will allow a contact to occur anywhere on the sphere, not just in some specific spot.) The location is further constrained by optionally specifying a contact "direction" in the local cross of axes of the segment or object (Figure 5.3). The "point" register contains the actual fixed surface contact point or the (variable) sliding surface point found during contact achievement, subject to the constraints in the other registers.

Achievement of a contact involves two "places" and a relationship. If the virtual contacts have any locational freedom, then the points actually used are those representing any closest pair of points satisfying the constraints on the two virtual contact sources. The "direction" registers (when non-nil) define the desired alignment of the virtual contact points (Figure 5.4). The monitor achieves the alignment by generating suitable facing instructions for the joint processors responsible for these contacts (Section 5.2.1).

The remaining degree of freedom is the distance between the contact "places", and this is described by an "upper limit" and "lower limit" distance range. The "initial distance" and "current distance" between the contact "places" are also stored (Figure 5.5). The distance tolerance is set from defaults associated with the "kind" field of the contact sign: both limits are zero for "touch" or "support", both are small positive values for "near", and the "lower limit" is zero and the "upper limit" is some maximum possible distance for "relate". A "surround" modifier is sep-

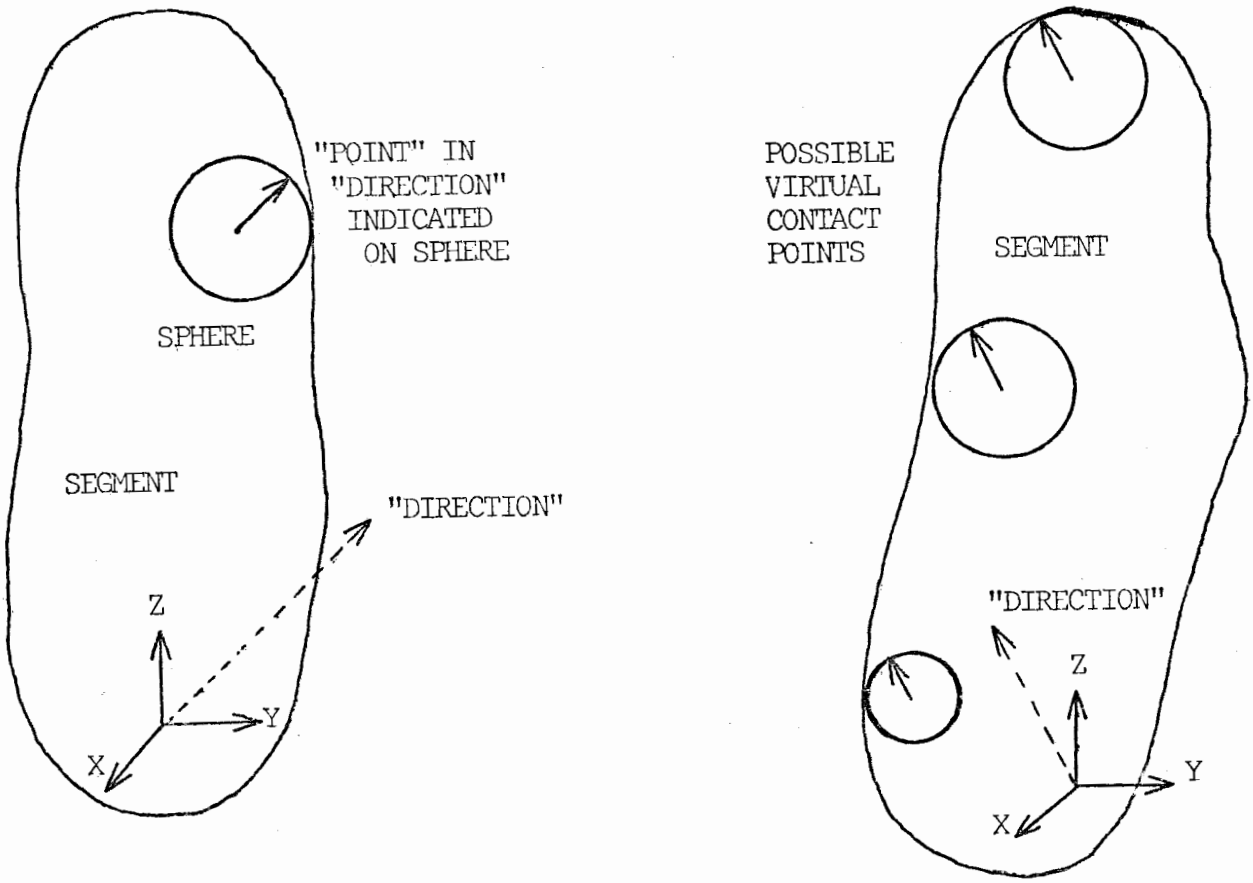


Figure 5.3 Virtual contact points.

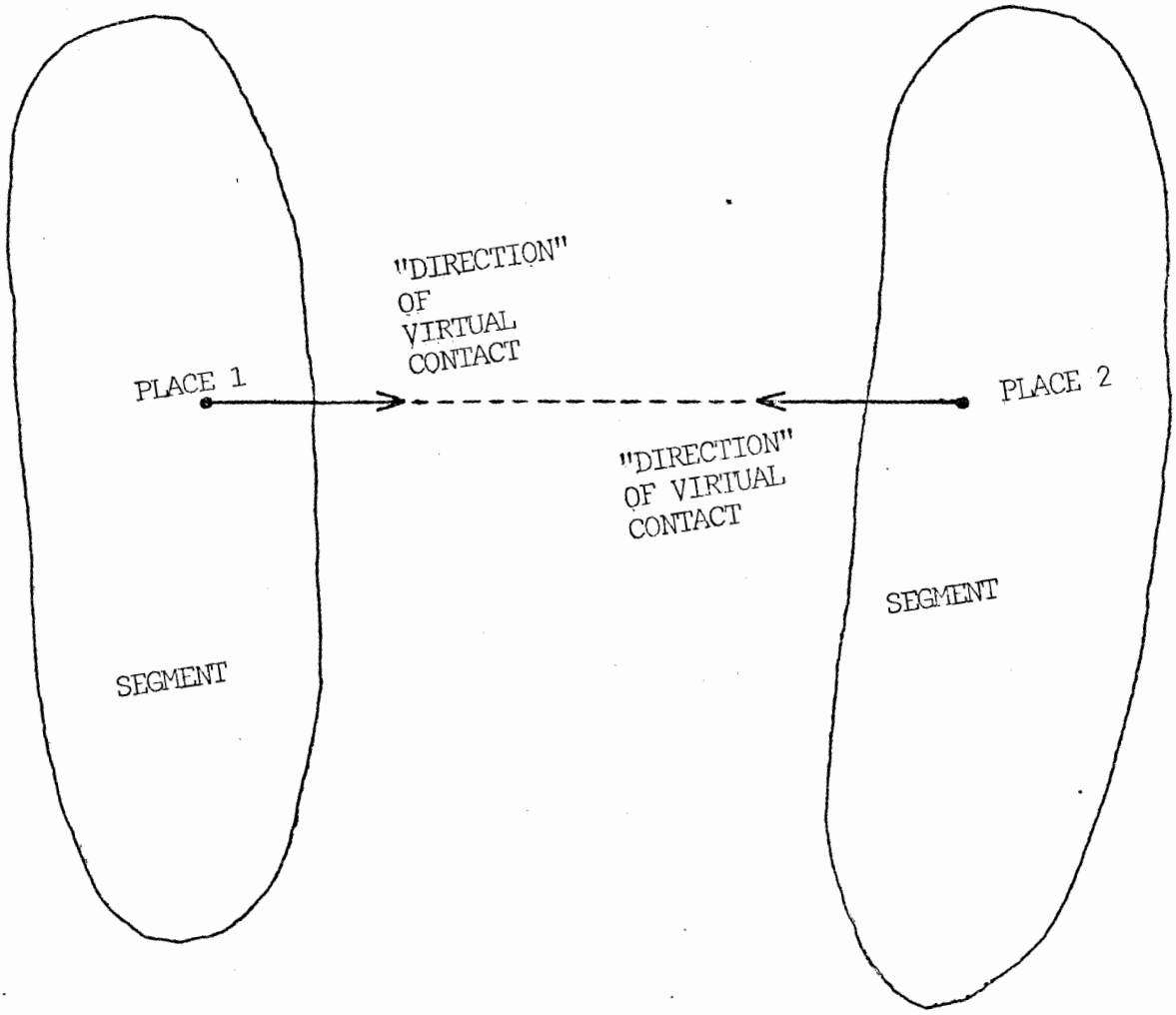


Figure 5.4 Alignment of virtual contacts.

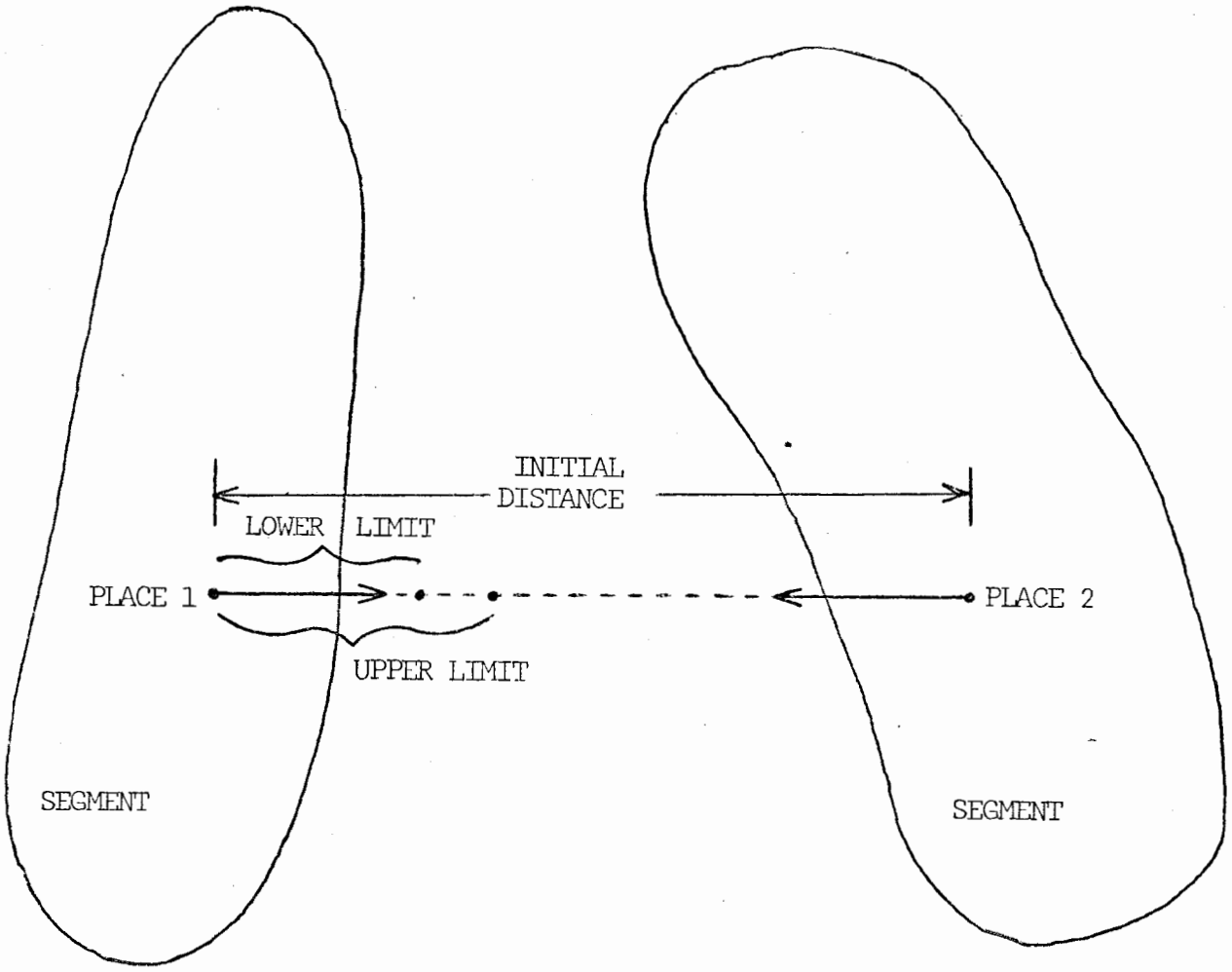


Figure 5.5 Distances involved in contacts ("near" illustrated).

arated into several contact blocks, each of which will be achieved independently but concurrently. Finally, "support" contacts indicate the weight they currently bear, since the monitor obtains this information when computing the body center of gravity from the individual segment positions. Contacts which are not expected to be supports have a value of nil in the "weight" register.

5.2.1 Determining contact scope

The monitor must determine which joint processors are to implement each contact. Since at least one virtual contact must always move toward the other by explicit effort of a joint processor, it is designated as (active) "contact 1". The other virtual contact ("contact 2") may or may not be actively moved as part of the contact relationship (indicated by the "role" register). The delegate joint processors for each active contact are chosen by examining the set of instructions preceding the contact time: a delegate is a joint processor having the direction sign with the latest starting time prior to the contact, with its duration containing the contact time, and also with the virtual contact point in its augmented scope. If there is more than one joint processor and direction pair satisfying these conditions for a virtual contact point, then that used is the one whose associated joint lies closest to the segment containing the virtual contact and whose augmented scope joint lies furthest from it.

If there are no such processors, then a direction instruction is generated for the distal joint of the segment containing the virtual contact. The default fixed end for that processor becomes the fixed end of the instruction. The duration is the single simulation interval prior to the contact. The instruction is interpreted as a movement of the virtual contact point (not the joint) to the zero vector in a reference system with

origin at the location of the other virtual contact's lower limit along the contact direction (Figure 5.6). (How this is done is described in the next section.) Once the direction instructions are determined, their "contacts" registers are set to refer to the appropriate contact block.

Any difference between the contact directions is adjusted by generating facing instructions for each active delegate joint processor. These will have the same fixed ends and durations as their respective direction instructions. The "area" facing register holds the contact direction, while the "goal" is the other virtual contact point (Section 4.1.3).

If both passive and active joint processors are involved in a contact, then the monitor will perform any movements of the passive joint first (subject to the established priority order). The active joint will then have the role of "pursuing" the passive one. If both segments are active in the contact, then each is given its turn at achieving the relationship. Since either could reach its goal first, there is some nondeterminism in this procedure; but this should only cause unusual results when the simulation interval is inordinately large.

5.2.2 Contact implementation

We have reduced the problem of achieving a set of arbitrary contacts to the problem of defining how a single contact modifier affects the execution of a direction sign. If a single direction sign has more than one modifying contact block, the movement of the contact point is the vector average of the set of contributing contact displacements which we shall define below. Multiple contacts may be achieved by a joint processor even if they occur at different times within the same specified instruction. If one such contact is not achieved to the specified tolerance, an error is reported.

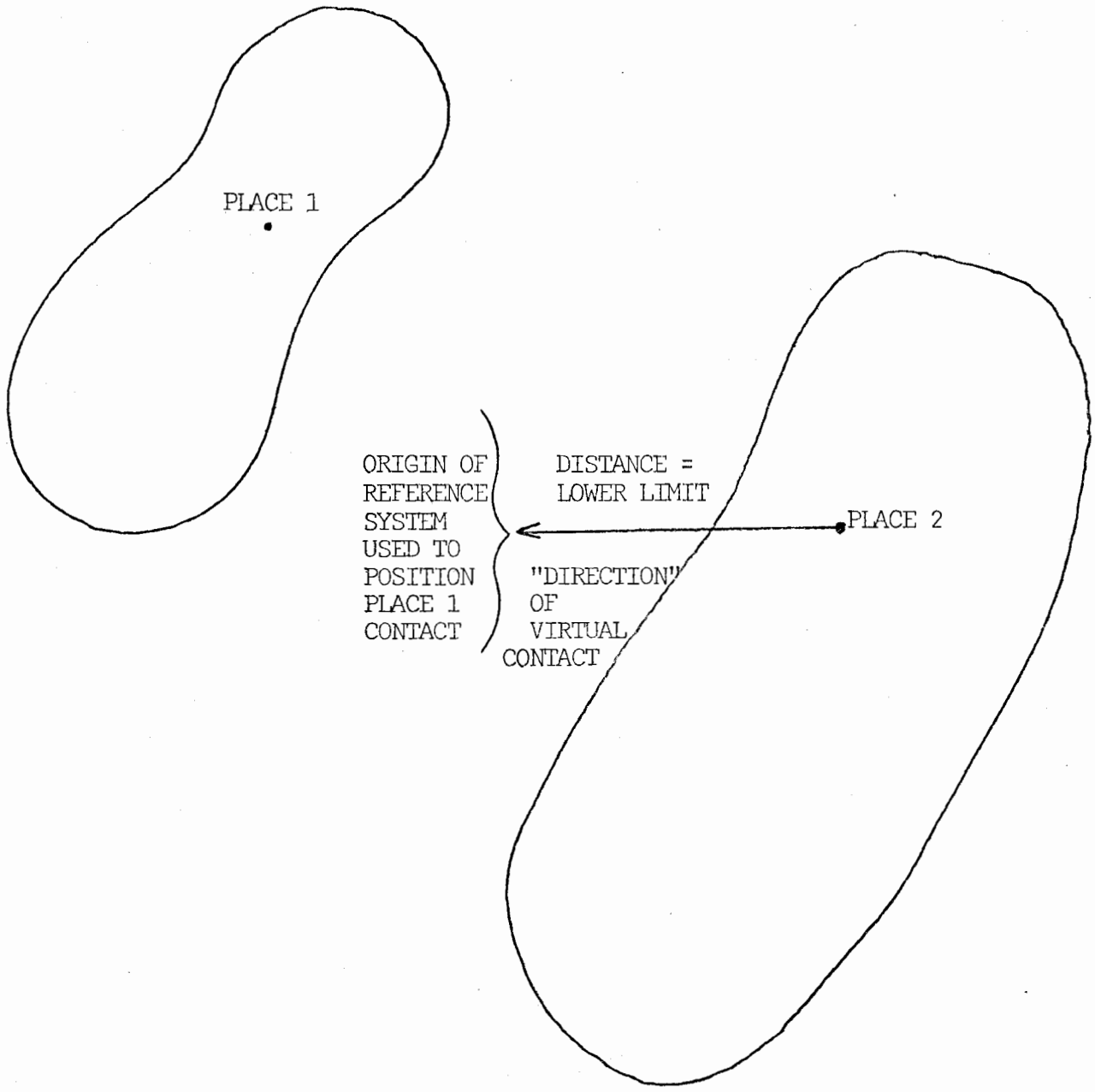
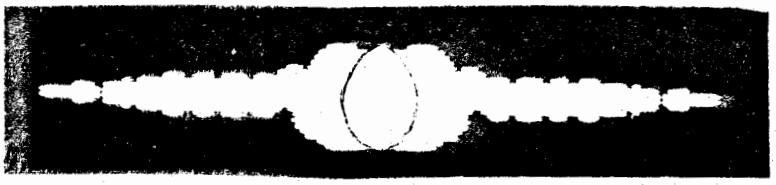


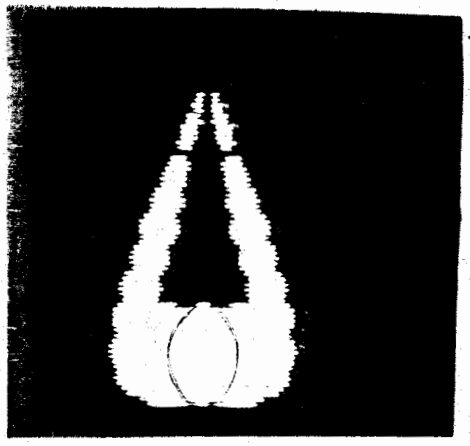
Figure 5.6 Goal of moving contact point.

By the joint selection criteria described in the preceding section, a delegate joint processor should approach the general vicinity of the contact; thus the contact will modify the normal path of the joint. The desired relationship is effected by altering the normal joint paths by two factors: one, the distance influence, to push or pull the virtual contact points toward or away from one another; the second, the time influence, to postpone the maximum effect of one point on the other until the very end of the direction sign duration. For example, consider forward middle movements of each arm from side middle positions (Figure 5.7a), modified at the end by a hand clap contact (Figure 5.7b). The first factor will insure that the hands continue beyond the forward middle position to achieve the contact in the saggital plane of the body. The second factor causes the additional movement to occur primarily at the end of the direction movement for each arm, otherwise the hands would approach one another too quickly and tend to cause contractions at the elbows (Figure 5.7, c and d).

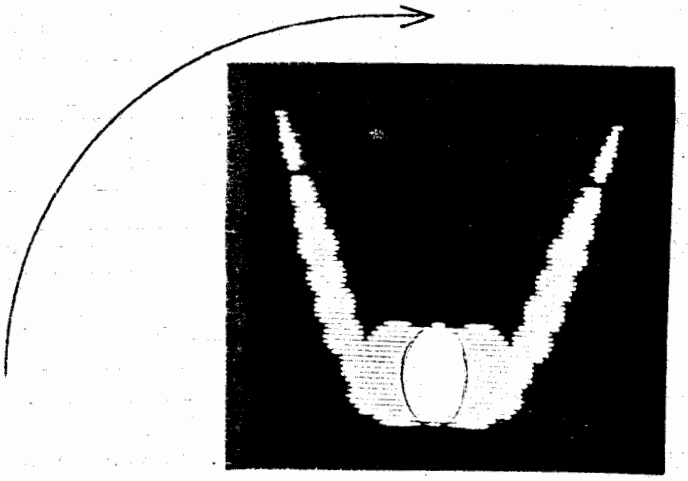
The distance influence has been chosen to be the piecewise linear function shown in Figure 5.8. The closer the virtual contact points, the more they affect each other's position. If the two points should move further apart than their initial distance (and this were not desired), then the distance influence remains at some minimal level. If the contact is still required, the points will tend toward each other no matter how far apart they get. On the other hand, if no other contact sign explicitly cancels a contact once achieved, the points are allowed to drift away naturally when moved by other instructions. The time influence is simply a linear function which is zero when the contact is first encountered as a modifier in a direction instruction and one when the contact is expected



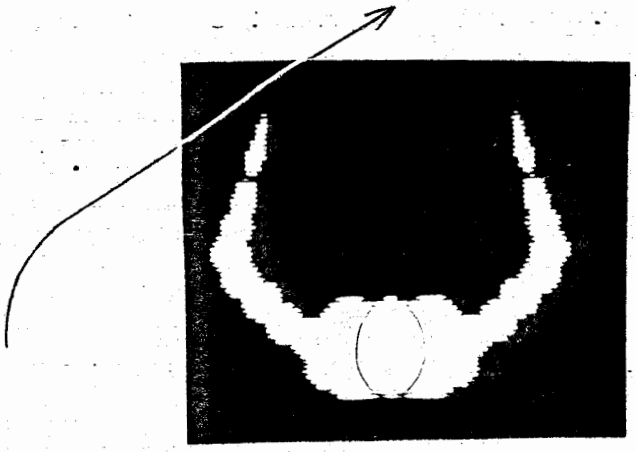
(a) start



(b) finish



(c) correct (curved path)



(d) incorrect (deformed path)

Figure 5.7 Hand clap modifying forward middle positions for both arms.

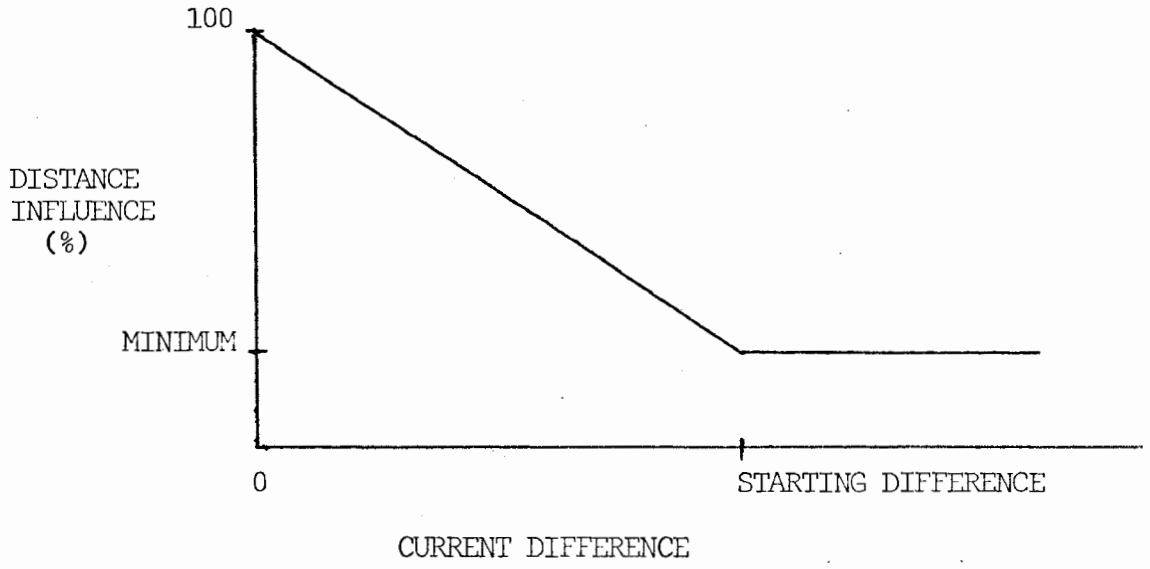


Figure 5.8 Distance influence.

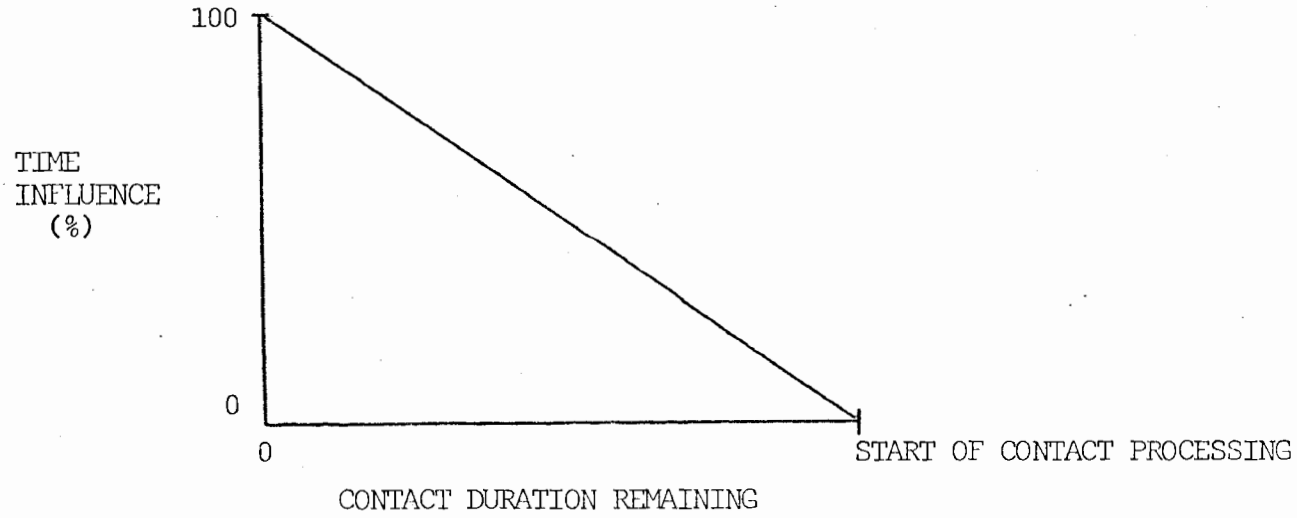


Figure 5.9 Time influence (for each contact point).

(Figure 5.9). The time influence value is undefined after the contact is achieved.

The two weighting factors are combined into a single influence value by multiplying them together and associating half of the resulting weight with each contact, even if only one is active (otherwise the active point approaches the passive point too quickly). Thus each delegate joint processor is responsible for achieving half the contact relationship; a passive contact does not participate at all. Since one active processor must move to actually achieve the desired relationship at the contact time, its influence is set to one on the very last cycle, otherwise the virtual contact points would still be in a state of "approaching the relationship."

Finally, the influence value is used to scale the distance between the two contacts (the contact displacement) so that they approach or repel one another. In each simulation cycle, the virtual contact is first (roughly) positioned by the direction instruction to the delegate joint processor. Then the contact displacement shifts the point position, and the monitor updates the body data base accordingly.

The timing information needed to process a contact is shown diagrammatically in Figure 5.10. The direction duration is the given duration of the direction sign chosen to implement the contact. The contact "duration interval" registers are assigned the difference between the beginning of the direction sign and the contact time for each delegate joint processor. (Thus the contact duration interval is less than or equal to the direction sign duration.) The "duration remaining" register of the contact block contains the difference between the current simulator time and the contact time.

The movement of the delegate joint processor for each contact can now

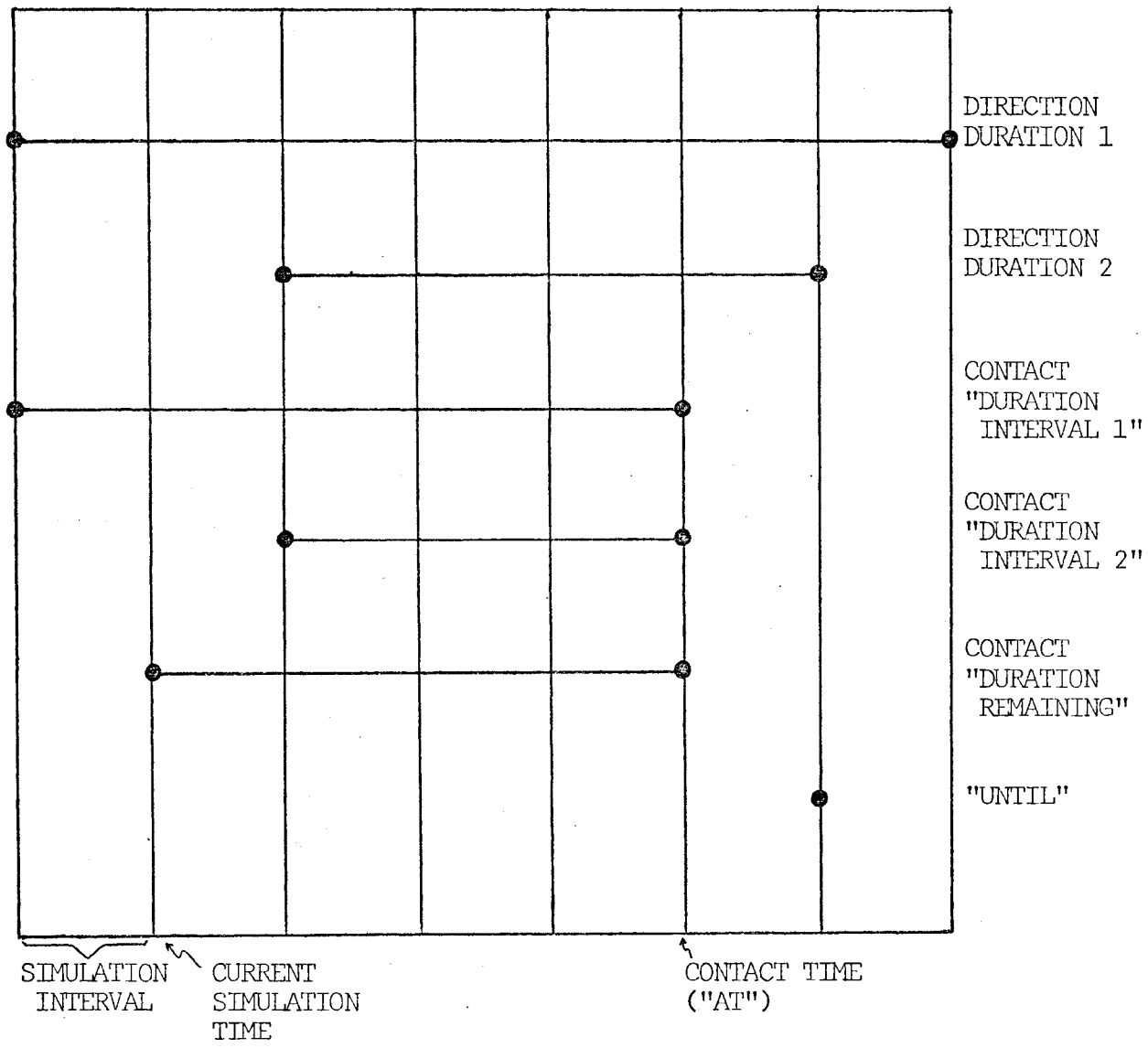


Figure 5.10 Contact timing for delegate joint processors.

be computed in the reference system of the direction instruction. The (joint processor, instruction) pair implementing the contact is allowed to execute according to the usual priority order. The joint is moved according to the direction destination (modified by any deviations and bends); then the following additional steps are performed:

1. Set the "difference vector" register of the contact block to the vector between the two virtual contact points. (The facing instructions will insure that the actual contact directions line up if required.) The length of the difference vector is placed in the "current distance" register of the contact block.

2. Compute the distance influence (Figure 5.8), a measure of how strongly the two contact points will attract one another:

   starting difference = $\left|\text{initial distance} - \text{lower limit}\right|$;

   current difference = $\left|\text{current distance} - \text{lower limit}\right|$;

   distance influence =

   $$\max\left(1 - \min\left(\frac{\text{current difference}}{\text{starting difference}}, 1\right), \text{minimum influence}\right)$$

   where the minimum influence is some small positive number such as 0.01.

3. Compute the time influence (Figure 5.9), a measure of how the contact achievement will be distributed over the contact duration.

   time influence = 1 - (contact duration remaining /contact duration interval).

4. Since two contacts are involved, each is assumed to achieve half of the contact (even if one is passive). The value stored in the "influence" register is therefore:

   influence = (distance influence)(time influence)$^{\frac{1}{2}}$.

   During the final movement in the last cycle when the time influence is 1, the processor moving last has influence = 1.

5. Finally the contact displacement is computed by scaling the difference vector by the lower limit tolerance and the influence:

contact displacement =

$$\text{(difference vector)}\ \frac{\text{(current distance - lower limit)}}{\text{current distance}}\ \text{(influence)}$$

This may reverse the direction of the difference vector, but that only indicates that the contacts are to be moved apart, not together. The virtual contact point is then moved from its current position by the contact displacement.

In order to accomplish the last step, the joint processor substitutes the virtual contact point for its associated joint in the "current position" direction register. It then executes a straight or radial path movement to the position computed in Step 5, as if the joint itself were situated at the virtual contact point. The joint position in this configuration is then computed, and intermediate joint bends are adjusted as necessary. This process will additionally bring the contact points together precisely at the end of the contact duration interval.

5.2.3 Contact maintenance

Contact maintenance is controlled by the "until" register of the contact block. If this value should be the same as the time at which the contact is achieved, then the contact is transient. The contact block is deleted; and the direction instructions to the delegate joint processors implementing this contact are free to assume their original destinations (if they have any remaining duration).

If the contact is to be maintained the contact block is not deleted, but its "duration remaining" register is left at zero. It will continue to affect the direction instruction it modifies until either that instruction's

normal end or the contact "until" time. During this interval the maintenance process is exactly the same as that used to achieve the actual contact by the last processor in the final cycle; the contact displacement is evaluated with an influence of 100% for one delegate joint processor or, if there are two, then 50% for the first and 100% for the second.

When the direction instruction ends before the "until" time, the monitor must generate a new direction instruction to accommodate the contact block. This instruction specifies a movement direction of (0,0,0) in the (new) reference system situated at the other virtual contact point and parallel to the reference system of the containing segment or object. The direction duration is the difference between the current simulator time and the "until" value; instruction processing (Section 4.2.2) will now insure that the contact is maintained in each simulation cycle during this interval.

6. PROGRESSION PROCESSOR

The progression processor has three primary responsibilities: to control the movement of the whole body in space, to determine the path of the center of gravity of the whole body, and to maintain balance when necessary. The movements are described by instructions to joints which support the body, but these instructions are collected together into a single stream for the progression processor:

```
type progression stream = sequence progression instruction.

type progression instruction = (agent: (joint, whole body);
                                start; rational;
                                action: concurrency).
```

A "concurrency" has already been defined for joint processors (Section 4.1). Each substream of a concurrency has an associated program counter with the same semantics as that of a joint processor.

Because support movements depend upon succeeding instructions (perhaps to different support joints), the progression processor must determine a movement in advance of its actual starting time. The progression processor may dispatch any of these instructions to another joint processor; so the original instruction, as well as its interpretation into the appropriate set of joint registers, must be saved:

```
type progression registers = (supports: sequence support joint;
                              interpreted: joint registers).

type support joint = (present: sequence instruction;
                      begin: rational;
                      prepare: sequence instruction;
                      onset: rational;
                      between: (ground, step, jump)).
```

The "present" instructions have starting time "begin", while the "prepare" instructions are the next set with starting time "onset" not equal to "begin". The action in the interval between these two times is determined by the

"present" and "prepare" instructions. For example, if these registers contain direction signs such that the duration of the "present" causes its end to coincide with "onset", then the "between" action is "ground"; that is, that supporting joint will maintain its contact with the ground. Time gaps between direction signs result in a value of "step"; and if no other support joint has a direction sign during this gap, then the result is "jump".

6.1 Progression Implementation

We shall assume that the support joints are the two ankles to simplify the discussion. An instruction may cause local changes to the body center of gravity or more global changes requiring instructions to be sent to other processors. Effects will be described from simplest to most complex, to aid understanding.

A shape description of the position kind must describe the relative positions of support joints. This position is assumed by a process similar to that presented in Section 4.2.1. When the shape description is a movement, it defines the (approximate) path of the center of gravity. (Although the path is defined in three dimensions, it is convenient to assume that if all the z-coordinates of the path are zero, it is actually a projection of the path onto the floor.) The path is approximate because the center of gravity is allowed to oscillate about that path during movements. A reasonable rule is to require the midpoint of the path of the center of gravity during a step to lie on the global shape path (Figure 6.1) whenever the step direction does not coincide with the shape direction.

Facing signs for a support joint are passed on to their respective processor input streams. A facing sign for the whole body causes a whole body rotation (changing "stance" [12] and similar changes in facing to the

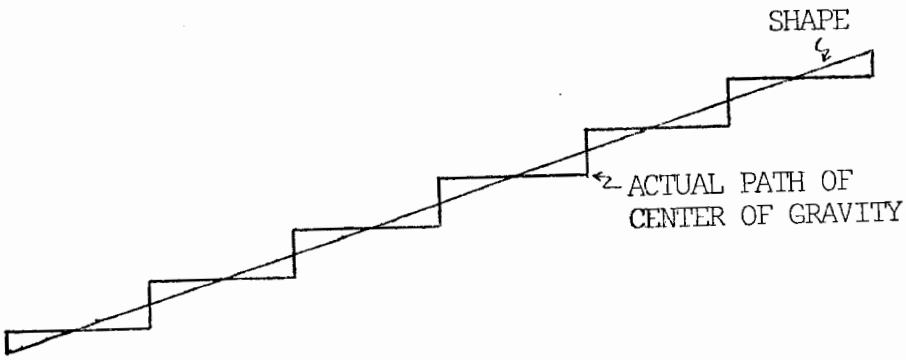


Figure 6.1 Approximating a shape description affecting support.

feet. These foot movements will appear as swivels (with friction) unless a simultaneous step is called for in the "between" register. Revolutions to the whole body change its orientation and introduce revolution signs to the support joints unless these already exist. Revolutions of the support joints, on the other hand, may induce a change in the orientation of the whole body depending on the current positions of those joints on the floor.

Direction signs provide the most concise means of specifying a complex movement and, as such, require that the instruction be expanded to include the significant preparation, propulsion, and "follow-through" stages of the movement. We have chosen to implement direction sign movements under the following assumptions:

1. The propulsion arises from a constant angular velocity at the fixed end. (Evidence for this assumption comes from biomechanical sources [6].)

2. The preparation phase is a fixed percentage of the previous direction sign [12].

Intermediate joint movements are constrained by the segment orientation limits, the fixed segment lengths, and the "boundary conditions" imposed by the segment positioned by assumption (1) and the geometry of the floor. Consider a leg, for example, which is to move forward from a starting position. During this movement we know the step length (hence the displacement of the center of gravity), the angle through which the upper leg must move to displace the center of gravity, , the time of toe lift-off and heel contact in the step, and the initial and final tilt angles at the knee. This information fully constrains the movement of the leg, provided that the center of gravity is forbidden from movements other than that required by the forward progression. The progression processor determines the contact times from the "present" and "prepare" direction instructions and generates contact

instructions for the monitor in advance. When combined with the direction instructions passed on to the ankle processors, smooth movement will result.

6.2 Balance

When the progression processor determines that balance is necessary during or at the end of a sequence of progression instructions it may adjust the body position. The progression processor bases its decision on the instructions it is preparing for and the number and geometry of current support points from the contact blocks in the monitor. If the number of contact points is zero, the body is in the air; and no adjustment is made. If there is one support, then whether or not an adjustment is made depends on the time until the next support instruction. (If this interval is too long, balance is necessary unless a new supporting joint is indicated: for example, falling from feet onto hands.) If there are three or more supports and the body center of gravity projects within the polygon formed by these points on the floor, then the body is presumed balanced. When this is not the case or when there are only two support points, then balance is established by rotating the center of gravity (in a non-forbidden direction) to bring its projection within the support line or polygon. The remainder of the body (that is, the set of parts not between the center of gravity and the supports) is simply displaced horizontally to the new position (subject to the separate forbidden vectors at individual joints).

7. CONCLUSIONS

In seeking a digital representation of human movement, an established movement notation system, Labanotation, has provided a wealth of well-structured information. The variety of human movement has been abstracted to five types of movement concepts; these form instructions which are interpreted by a simulator with knowledge of support requirements, body structure, and body surfaces. By designing the simulator as a network of communicating processes, we obtain very general and flexible control over individual joints, body segments, and the whole body.

Components of the simulator have been implemented in LISP and FORTRAN on a UNIVAC 90/70. While the simulator itself is not expected to produce graphic commands at a real-time rate, these commands will be stored in a file and interpreted in "batches" by the graphic display program. We expect that this process will be fast enough to animate the body model (drawn with circles or shaded disks to represent each sphere) on a graphics configuration consisting of a PDP-11/60 computer and a Vector General 3404 refresh display. Sequential snapshots may be produced on our Ramtek GX-100B color video display to obtain permanent video or film records of the solid figure in motion.

8. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the partial support of NSF Grant MCS76-19464. We also wish to thank Bill Wood, Jr. for his valuable suggestions which arose from an initial implementation of the simulator control structure, and John Fedak for his role in refining the movement instructions while implementing the Labanotation compiler.

9. REFERENCES

1. Badler, N.I., Temporal scene analysis: Conceptual descriptions of object movements, Report No. UP-MS-CIS-76-4, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA., (February 1975).

2. Badler, N.I., O'Rourke, J., and Toltzis, H., A human body modelling system for motion studies, Movement Project Report No. 13, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA (July 1978).

3. Badler, N.I., and Smoliar, S.W., Digital representations of human movement, Computing Surveys (to appear 1979).

4. Benesh, R., and Benesh, J., An Introduction to Benesh Dance Notation, A. & C. Black, London, 1956.

5. Brinch-Hansen, P., Operating System Principles, Prentice Hall, New York, 1973.

6. Cappozzo, A., Figura, F., Marchetti, M., and Pedotti, A., The interplay of muscular and external forces in human ambulation, J. Biomechanics 9 (1976) 35-43.

7. Catmull, E., A system for computer generated movies, in Proceedings of ACM Annual Conference, Vol. 1 (1972) 422-431.

8. Eshkol, N., and Wachmann, A., Movement Notation, Weidenfeld and Nicolson, London, 1958.

9. Fedak, J., An initial design specification of a syntactic analyzer for Labanotation, Movement Project Report No. 10, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA (January 1978).

10. Finkel, R., Taylor, R., Bolles, R., Paul, R., and Feldman, J., An overview of AL, a programming system for automation, in Proceedings of the Fourth International Joint Conference on Artificial Intelligence (August 1975) 758-765.

11. Hoare, C.A.R., Notes on data structuring, in Dahl, O.-J., Dijkstra, E.W., and Hoare, C.A.R. (Eds.), Structured Programming, Academic Press, New York, 1972.

12. Hutchinson, A., Labanotation, Theatre Arts Books, New York, 1970.

13. Jay, L., A stick-man notation, Dance Observer (January 1957) 7-8.

14. Laban, R., Choreutics, Ullman, L. (Ed.), Macdonald and Evans, London, 1966.

15. Lozano-Perez, T., The design of a mechanical assembly system, Report AI-TR-397, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA (December 1976).

16. Mezei, L., and Zivian, A., ARTA, an interactive animation system, in Proceedings IFIP Congress (1971), North Holland Pub., Amsterdam, 429-434.

17. O'Rourke, J., Three dimensional motion of a three link system, Movement Project Report No. 11, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA (June 1978).

18. Pikula, J., Sutton notates Bournonville, Dance Magazine (November 1975) 31.

19. Smoliar, S.W., A parallel processing model of musical structures, Report AI-TR-242, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA (1971).

20. Smoliar, S.W., and Tracton, W., A lexical analysis of Labanotation with an associated data structure, Proceedings of ACM Annual Conference (1978).

21. Smoliar, S.W., and Weber, L., Using the computer for a semantic representation of Labanotation, in Computing in the Humanities, Lusignan, S., and North, J.S. (Eds.), University of Waterloo Press, Waterloo, Ontario, 1977.

22. Spegel, M., Programming of mechanism motion, Report No. CRL-43, Division of Applied Science, New York University, New York, NY (November 1975).