

**A Graphics Editor  
for  
Benesh Movement Notation**

by

**Baldev Singh**

A thesis  
presented to the University of Waterloo  
in partial fulfillment of the  
requirements for the degree of  
Master of Mathematics  
in  
Computer Science

Waterloo, Ontario

September 1982

Copyright © Baldev Singh, 1982

To my parents  
for their encouragement and support  
for my studies in Canada

I certify that I have read this thesis and that in my opinion it is fully adequate in scope and quality, as a thesis for the degree of Master of Mathematics in Computer Science.



(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate in scope and quality, as a thesis for the degree of Master of Mathematics in Computer Science.



(Reader)

I certify that I have read this thesis and that in my opinion it is fully adequate in scope and quality, as a thesis for the degree of Master of Mathematics in Computer Science.



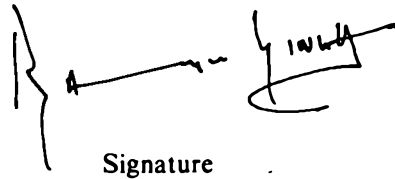
(Reader)

Approved for the University Committee on Graduate Studies

(Dean of Graduate Studies)

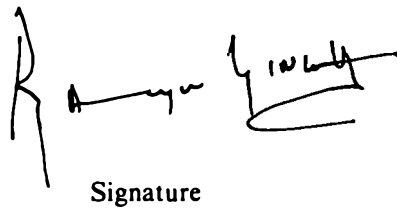
I hereby declare that I am the sole author of this thesis.

I hereby authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purposes of scholarly research.



Signature

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by any other means, in total or in part, at the request of other institutions or individuals for the purposes of scholarly research.



Signature

The University of Waterloo requires the signatures of all persons using or photocopying this thesis.  
Please sign below, and give address and date.

## Abstract

This thesis describes an interactive computerized editor for *Benesh Movement Notation* which aids in the preparation of dance scores using a medium resolution colour display. Benesh Movement Notation is a two-dimensional system for recording human movement in three dimensions of space. The Benesh notations has been successfully used in recording a wide repertoire of dances. The preparation and revision of scores is a lengthy and error-prone process which interactive editing techniques can greatly facilitate. The current state and future extensions of a prototype editing system for Benesh notation and its user interface are discussed.

## Acknowledgements

I am fortunate to have been associated with advisors who took an active interest in my work. Dr. John C. Beatty, whose research in Computer Graphics spurred my own interest, was a very helpful supervisor and guided me in trimming the topic to a manageable size for one thesis. Dr. Kellogg S. Booth, filling in as my supervisor in Dr. Beatty's absence, was a source of many useful suggestions, advice and encouragement. Prof. Rhonda Ryman of the Dance Group provided guidance on the Benesh Movement Notation and other dance related issues throughout the project.

Discussions with Prof. Benton Leong over daily dinners produced many helpful suggestions. Several members of the Computer Graphics Laboratory supplied useful tools developed in their projects. In particular, the graphics support software written by Paul Breslin was especially valuable. Steve MacKay's colour table manipulation routines and Darlene Plebon's tablet software saved considerable development time. This thesis was typeset on an Aps  $\mu$ -5 phototypesetter using software developed by Richard Beach.

It is important in a project such as this to elicit the evaluations of skilled users about the ease and naturalness with which the system can be used. In this regard, I would like to thank Robyn Hughes, choreologist with the National Ballet of Canada, Sandy Caverly Lowery and Joan Mallet of York University, Wendy Walker, choreologist with the American Ballet Theatre, and the dancers who attended the First International Summer School in Benesh Movement Notation held at the University of Waterloo in August 1982. I am especially grateful to Monica Parker, director of the Institute of Choreology, London, England for her support of the project.

I would also like to thank my officemates Susan Goetz and Doris Kochanek who have been patient listeners and offered lots of advice. I am especially grateful to Doris for an untold amount of encouragement and for a thorough reading of the manuscript which resulted in a number of improvements and clarifications.

The Computer Science Department and the Natural Sciences and Engineering Research Council provided much appreciated financial support and facilities on which this project was carried out. I have also benefited from many discussions with Mert Cramer and Dr. Tom Calvert of Simon Fraser University. Finally, I am grateful to all members of the Computer Graphics Laboratory for providing a pleasant environment in which this work could thrive.

## Table of Contents

1. INTRODUCTION	1
2. PROBLEM DESCRIPTION	4
3. THE BENESH MOVEMENT NOTATION	6
4. EDITOR	18
4.1. Data Structure Manipulator	19
4.1.1. Manipulation of Frames	20
4.1.2. Manipulating Symbol Positions within a Frame	21
4.1.3. Construction of Benesh Symbols	22
4.2. Display Handler	23
4.2.1. Frame Display	23
4.2.2. Menu Display	25
4.3. Command Loop	25
4.3.1. Error Recovery	26
4.3.2. Abort and Undo	27
5. THE USER INTERFACE	29
5.1. Hardware and Software	30
5.2. Interaction Language	34
5.2.1. The Conceptual Model	36
5.2.2. Recognition versus Recall	37
5.2.3. The Command Set	38
5.2.3.1. The Menu Structure	40
5.2.3.2. Spatial Resolution and Visual Discrimination	47
5.2.3.3. Text versus Iconic Labels for Menu Items	48
5.2.3.4. Dynamic versus Static Menus	49
5.2.4. Consistency	50
5.2.5. Simplicity	51
5.3. Display Representation	52
5.3.1. Feedback	52
5.3.2. The System State	56
5.3.3. The Current Status of the Score	56
6. FUTURE EXTENSIONS	59
6.1. Extension to Editing Functions	59
6.2. Movement Verification	62
6.3. Hard Copy Facilities	69
6.4. Improvements to the User Interface	69
7. CONCLUSIONS	72
References	73
Appendix.A: List of notation systems	83
Appendix.B: A sample Benesh score	86
Appendix.C: Finite state diagrams of the editor	90



## List of Illustrations

1.1.	Structure of Benesh Editor	3
3.1.	Human body in anatomic position	7
3.2.	The Benesh stave	8
3.3.	Use of basic signs	10
3.4.	Various head positions	11
3.5.	Main body signs	12
3.6.	Movement lines	13
3.7.	Jumps	14
3.8.	Changes in effort	15
3.9.	Time signature and bars	17
4.1.	Linked list storage structure for Benesh score	19
4.2.	Frame node	22
4.3.	Display window	24
5.1.	Ikona's video chain	31
5.2.	Foreground and background planes	32
5.3.	Display layout of information in the background plane	33
5.4.	Display screen with tablet and puck	35
5.5.	Tablet puck	39
5.6.	Root menu	41
5.7.	Frame menu	42
5.8.	Selecting an edit frame	43
5.9.	Selecting an insertion position	44
5.10.	Selecting a frame sequence	45
5.11.	Window for constructing main body symbols	46
5.12.	Standard architectural symbols [Dreyfuss 72]	47
5.13.	Menu of direction symbols	47
5.14.	Body menu	49
5.15.	Different levels of feedback [Foley 82b]	53
5.16.	Icons for lexical feedback	53
5.17.	Alternate set of icons for lexical feedback	54
5.18.	Syntactic feedback	55
5.19.	Buddha icon [Dreyfuss 72]	55
5.20.	Tracker icons	57
5.21.	Status information display	58
6.1.	Scroll bar	60
6.2.	The skeleton	63
6.3.	Measure of average men and women [Dreyfuss 59]	66
6.4.	Ambiguity in extremity positioning	66
6.5.	Main body symbol	67
6.6.	Human body representations using solids	68
6.7.	Scrolling menu	70

C.1.	State diagram of the editor	91
C.2.	QUIT command	92
C.3.	ADD_FRAME command	93
C.4.	EDIT frame command	94
C.5.	DELETE frames command	95
C.6.	MOVE frames command	96
C.7.	COPY frames command	97
C.8.	SAVE frames command	98
C.9.	PUT frames command	99
C.10.	ARCHIVE score command	100
C.11.	DE_ARCHIVE command	101
C.12.	Editing a frame	102
C.13.	Main body part positioning	105
C.14.	Body limb positioning	106
C.15.	ERASE_SYMBol command	107
C.16.	MOVE_SYMBol command	108
C.17.	TEMPO command	109
C.18.	TIME_SIGnature command	110
C.19.	RHYTHM command	111
C.20.	DIRECTION command	112
C.21.	NOTES command	113
C.22.	CLEAR working frame command	114
C.23.	BAR_LINE command	115
C.24.	NEW_FRAME command	116

## 1. INTRODUCTION

*"Literature does not survive by the spoken word. Music does not survive by memory alone. Dance without notation is a transient art, doomed to precarious existence and an early oblivion."*

*- A. Hutchinson*

Notation plays an important role in today's technological society. It acts as a communication medium between inventors and those who will implement their ideas. Without notation, knowledge from the past can only be based on oral transmission as recorded in human memory. Because human memory is fallible, the knowledge thus transmitted from person to person is relatively limited and often inaccurate. Any comparative analysis of past and present works is virtually impossible, and thus the depth and breadth of anyone's work is limited to what can be carried in his memory.

In most fields of human study we take for granted the existence of a precise and concise notation. An example is the *alphabet* one of man's greatest inventions without which our culture as we know it today would be unimaginable. However, the existence of such a universal notation is not true in the field of human movement. The problems mentioned above are especially apparent in *ballet* - a classical style of expressive dancing. Most ballets from the past have been completely lost, leaving behind only the memories of the few people fortunate enough to have witnessed them. The losses have been enormous even among the most recent ballets [Hall 64, 65].

*Choreography* is the art of composing dance steps and sequences, such as for a ballet. By the standards of other theatrical arts such as opera and drama, the choreographic heritage is very poor. The problem is that choreography suffers in transmission. Dancers working without a choreographer's guidance tend to change the choreography, moulding the lines and rhythms into patterns that are comfortable for them to perform. When a role is handed over, usually by one dancer teaching it to another, further changes take place. The *ballet master* then works on the results of this transmission, trying to correct anything that seems out of place because it is not properly danced or simply because it is untrue to the original. This frequently introduces further changes. All these changes are cumulative and often so untrue to the creator's intentions that a few years later the choreographer can hardly recognize his own creation. The few works that do survive are ones that suit the widely varying tastes of dancers from several generations, or are easy to perform [Hall 65].

To minimize these losses from imperfections in human transmission, a concise and economical notation is needed for recording human movements. This need has been recognized for a very long time and during the past five centuries many attempts have been made at devising a suitable notation. It is believed that the ancient Egyptians made use of *hieroglyphics* to notate dance and that the Romans employed a method for recording salutatory gestures [Encyc. Brit. 74]. But the earliest attempts that are documented precisely date from the second half of the 15th century: these are manuscripts of Spanish, Italian and Burgundian origins which used abbreviations to indicate the familiar steps of the *basse dance*, the foremost social dance of the early Renaissance.

As professional ballet became distinguished from folk dance, many systems of movement notation emerged for ballet. Appendix A lists a few of the known movement notation systems in chronological order of their development. Some of these systems marked the steps of each dancer with lines superimposed over a sketch of the floor plan, while others used small skeleton figures drawn in dance positions. Still others adapted musical notation, using the conventional symbols to represent space instead of pitch. There have been many individualized systems. But most of these failed to be practical under rigorous professional conditions. Among the few that have invited substantial professional interest are *Labanotation*, developed by Rudolf Laban in 1928 [Laban 75; Hutchinson 77], and *Benesh notation*, a system invented by Rudolf Benesh in 1947 [Benesh 56, 77; Causley 67].

Human movement is very complex; for this reason movement notation systems are inherently complex and difficult to master. Using computer technology it is possible to provide tools that simplify the use of these notations. Projects for developing such tools to interpret movement notations are in progress at the University of Pennsylvania [Badler 78a, 78b, 79a, 79b, 80; Brown 76a, 76b, 78; Smoliar 77, 78; Fedak 78], the University of Waterloo [Savage 77a, 77b], the University of Iowa [Sealey 81], the Simon Fraser University [Calvert 78, 80, 82a, 82b, 82c], Sydney University [Herbison-Evans 74, 78, 79a, 79b; McNair 79, 80] and elsewhere [Lansdown 77, 78]. With the exception of the Sydney University project which uses Benesh notation, almost all of these projects deal exclusively with the interpretation of Labanotation.

The Computer Graphics Laboratory at the University of Waterloo, in cooperation with the Dance Group and the National Ballet of Canada, are currently investigating the computer editing of Benesh Movement Notation. The aim is to provide state-of-the-art computer tools for creating, editing, and verifying records of human movement. This thesis describes a first step in this direction: the design and implementation of a graphical editor based on the Benesh Movement Notation. A prototype based on the ideas presented here has been successfully implemented and tested by professional choreologists. The structure of this prototype editor is shown in figure 1.1. Chapter 2 explains the problem and those aspects explored in this thesis. The basics of Benesh Movement Notation are outlined in Chapter 3. Chapter 4 discusses the internal workings of the editor while Chapter 5 describes the editor's user

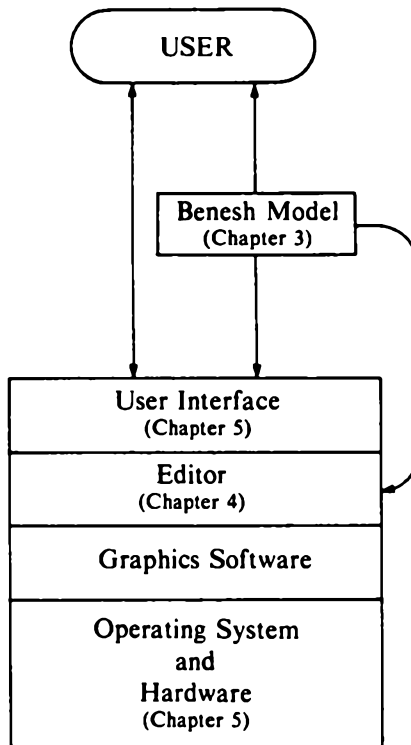


Figure 1.1. Structure of the Benesh Editor.

interface. Suggestions for future extensions to the editor are discussed in Chapter 6. Some observations and concluding remarks are presented in Chapter 7. Finally, a videotape illustrating the use of the prototype editor accompanies this thesis. A copy of this videotape can be obtained from the Computer Graphics Laboratory, University of Waterloo.

## 2. PROBLEM DESCRIPTION

*"It is essential to release humanity from the false fixations of yesterday, which seem now to bind it to a rationale of action leading only to extinction."*

*- R.B. Fuller*

At first glance, many movement notations seem similar to musical notations. Although both notations serve as a communication medium between composers and performers, they are used differently. Musicians learn a composition from musical manuscripts, whereas dancers learn their parts directly from the *choreographer* (a dance composer) through oral communication. Thus creating dance requires direct interaction between the choreographer and the dancers. In music, notation is the means by which a performer realizes the creator's conception of a composition [Brown 78]. In contrast, movement notations are used to record the finished product for historical preservation. However, they can also be used for comparative analysis and reconstruction at a later date without the choreographer's presence or the need to repeat the entire interactive process. While music notations are well standardized, dance notations are still being developed. The editor described in this thesis is based on the Benesh Movement Notation, a system which is widely used by professional dance companies. A library of ballet scores written in Benesh notation is maintained by the Institute of Choreology in London (England), and a number of ballets have been successfully reconstructed using these scores.

The process of recording dance in Benesh Movement Notation begins with a *choreologist* (a specialist in Benesh notation) attending rehearsals and making rough notes which are gradually refined during subsequent rehearsals. This score is later checked for errors in the recorded movements and in the notation usage. Once the score has been checked, it is put into its finished form by an *autographer*, who lays out pages and works with special drafting pens. Upon completion of this process the score is ready for printing.

During rehearsals, dancers learn complete movements with all parts of their body operating in parallel. However, the notation must record all changes in each body part individually. If there are many dancers quickly learning complex movements, it becomes difficult to capture all the details. The recording process is limited by the speed at which the choreologist can write. Therefore the choreol-

ogist usually notes only the essential key positions, filling in the details at later rehearsals. But in a large ballet company, where many hours each day are devoted to teaching new steps, the choreologist works with different groups of dancers throughout a ten to twelve hour period. This makes filling in details after rehearsals virtually impossible. The task becomes even more difficult when the choreographer decides to add or drop parts of the dance or revise steps and sections. Thus the choreologist has to constantly struggle with the organization of notes, filling in the missing details during *cleanup rehearsals* conducted after all the dancers have learned their parts.

Under these conditions the choreologist is often unable to work on the final draft of the score as the rough notes accumulate. There is just not enough time until after the dance is in performance. Then, many hours must be spent laboriously copying the rough notes, laying out the pages, and refining the notation.

It should be apparent that the basic problem facing a choreologist is one of information overload which worsens from rehearsal to rehearsal. Any improvements to this process require a faster means for recording movements (ideally at the speed of performance), an efficient mechanism for storing and retrieving this information for editing or verification, and means for producing the finished score in hardcopy form.

One possible approach would be to store the rough score in a computer system by scanning it with a video input device at the end of each rehearsal. The time between rehearsals could be devoted to editing and verifying this information through a graphical editor and producing a medium quality hard-copy output that could be used for further refinements at subsequent rehearsals. Alternatively, movement information could be directly entered into a computer system during the rehearsal itself using goniometers [Calvert 80, 82a; Townsend 77] or three-dimensional imaging techniques [Pierrynowski 80, 82; MacKay 82a]. *Goniometers* are devices which measure angles between body parts. Another possible method is the use of *electromyography* which involves the study of electric currents set up in muscle fibres by body movements [Grieve 76]. The information thus gathered could be mapped into any notational system. Calvert has successfully used goniometers as input devices to produce Labanotation scores of the dancer's movements [Calvert 82a]. The mapped information could be updated using a computer editor and checked for notation usage and correctness of the recorded movements. Finally, high quality hard-copy could be produced for publishing or use at later rehearsals.

Although substantial progress has been made in recording and analyzing movements, tools for editing scores, especially in Benesh Movement Notation, are virtually non-existent. Much more research is needed to make editing movement scores as easy and convenient as editing text. The work presented in this thesis deals with the issues involved in editing dance scores based on the Benesh Movement Notation system. A similar editor based on Labanotation has been developed at the University of Pennsylvania [Brown 76a, 76b].

### 3. THE BENESH MOVEMENT NOTATION

*"Science has dual history. It is a tale of the birth of men with great powers of abstraction, and it is a story of the evolution of languages peculiarly adapted to these abstractions"*

*- H. Levy*

The basic problem in any movement notation is that of coping with the enormous amount of information needed to record movements of each individual body part in three dimensions of space and in time. One solution is to use symbols. But this would require a very large number of symbols, and would result in a cumbersome notation. For a notational system to be workable, it must be as simple and legible as an alphabet. But we find that completeness and accuracy seems to make demands that are incompatible with simplicity and legibility. Some of the essential details a movement notation must capture include:

- the location of the body in the movement arena and in relation to other performers;
- the direction in which the body is facing with respect to the observer;
- the exact position of body limbs;
- the details of head, torso and pelvis;
- movement quality or dynamics;
- the manipulation of props.

Since a movement notation must record movement as it is actually seen, we need to establish a viewpoint. The standard choices for such a viewpoint are that of the dancer and that of the observer; each is a reflection of the other.

For uniformity in body description the *anatomic reference system* is used. The body in the anatomic position is erect, facing forward with arms at the sides, as shown in Figure 3.1. Henceforth all body positions will be discussed with respect to the following planes through the body:

*Mid-sagittal* - the plane vertically dividing the body through the mid-line into *right* and *left* halves.



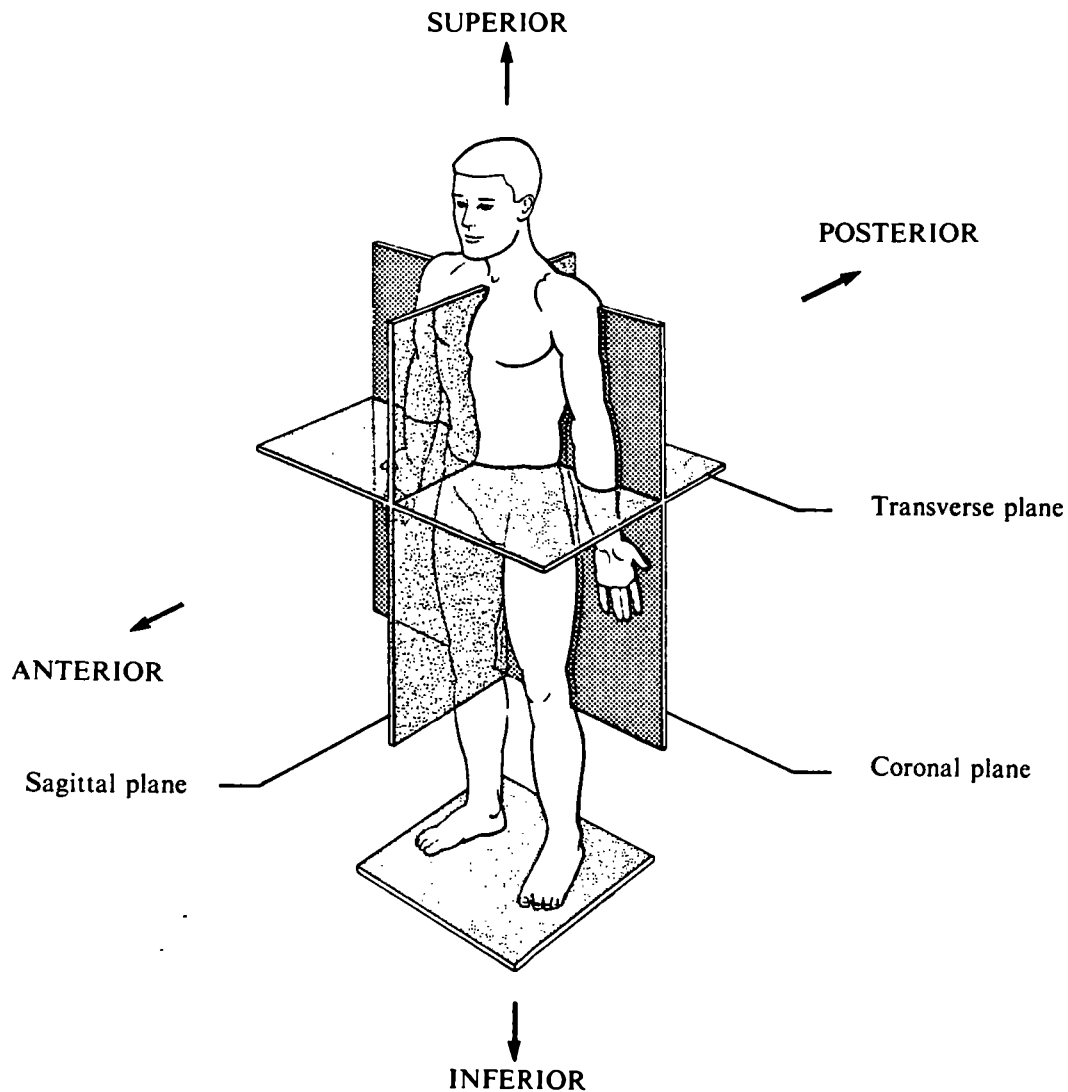


Figure 3.1. Human body in anatomic position.

*Sagittal* - any plane parallel to the mid-sagittal plane vertically dividing the body into *right* and *left* portions.

*Transverse (Horizontal)* - any plane dividing the body into *superior* (upper-most or above; e.g. the head is superior to the neck) and *inferior* (lower-most or below; the foot is inferior to the ankle) portions.

*Coronal (Frontal)* - any plane dividing the body into *anterior* (toward the front) and the *posterior* (toward the back) portions at right angles to the sagittal plane.

Benesh Movement Notation, invented in 1947 by Rudolf Benesh, records movements using marks on a matrix representing a human figure. The five line music *stave* serves conveniently as a matrix dividing the human body at the top of the head, the shoulders, waist, knees and the feet as shown in Figure 3.2. Two horizontal dashed lines known as *leger lines* have been added to cover all possible body positions. The leger line above the stave lines represents the maximum reach of the hands when the arms are fully extended upwards. The leger line below the stave lines is a *reflection* of the maximum point reachable by the feet during a jump from the standing position. The use of the second leger line will be explained further in discussion of methods of recording jumps and similar movements. Since the span of horizontally extended legs is approximately equal to the body height with arms fully extended above, all possible body positions can be enclosed in a square. So the stave is divided into square *frames*, each representing a body position [Grater 82]. All this makes the notation visual and avoids the need for a large number of symbols. The leger lines are not generally shown in Benesh notation records, but are always implied. Also implicit is the vertical dashed line, known as the *body mid-line*, which divides the body into left and right halves. The body mid-line is not the *line of gravity*, but does coincide with it whenever the body is in its anatomic position (see Figure 3.1).

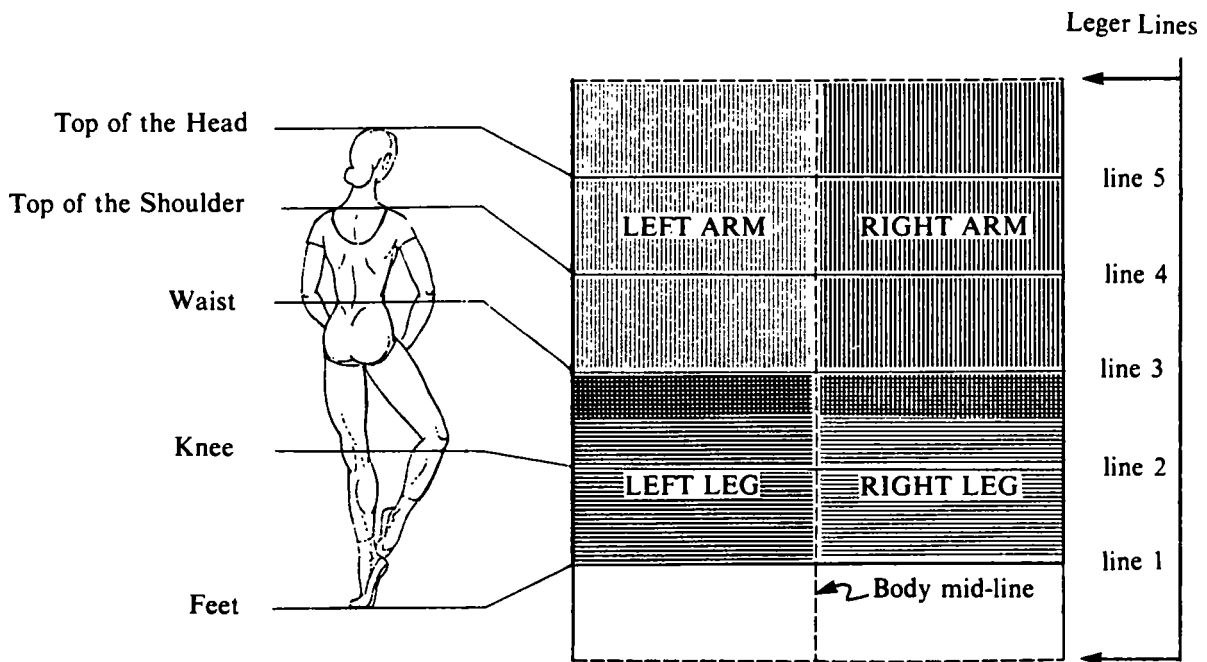


Figure 3.2. The Benesh stave.

NOTE: The doubly hashed region is the overlap for hands and feet.

The viewpoint used in Benesh notation is that of the performer. All movements are recorded as observed from behind the performer. Basic body positions are recorded by noting the projection of the four *body extremities* (hands and feet) and the *bends* (in knees and elbows) onto the coronal plane. The projection onto the coronal plane is not a true projection of the whole body figure. Rather, it is the projection of the body segments in relation to the body mid-line onto the corresponding areas between the stave lines. The projections provide two dimensional information. For three dimensional information all that is necessary is to know whether a limb is in front of or behind the frontal plane. This is recorded using the basic signs:

- level (with the coronal plane)
- ! in front (of coronal plane)
- behind (the coronal plane)

However, it is not necessary to state how far in front or how far behind the extremities are. Only one place is physically possible because limbs are of fixed length and are attached to the body at specific locations. The basic signs for bends in knees and elbows are:

- + level (with the coronal plane)
- ⊕ in front (of coronal plane)
- ⊗ behind (the coronal plane)

Again, it is not necessary to say how bent a limb is, as this is governed by the position of the corresponding extremity and the knee or elbow. Figure 3.3 illustrates the use of these basic signs.

The signs for all limbs are identical and thus do not identify the limb (right or left arm or leg) to which they refer. Instead this is inferred from the location of the sign. Figure 3.2 shows the *domains* for the four body limbs. Whenever a limb sign moves out of its domain, it is lightly crossed out with a diagonal stroke known as a *crossover*. There are two kinds of crossovers: a *lateral crossover* in which an extremity or bend (knee or elbow) moves over to the opposite side of the body; and a *vertical crossover* in which a foot or bent knee moves above the waist line, or a hand or bent elbow moves below the waist line. The two crossover signs are:

- ↙ lateral crossover
- ↘ vertical crossover

However, crossovers are not needed when the context clearly specifies the particular body limb. For example, when the feet are below the hands as shown in Figure 3.3.

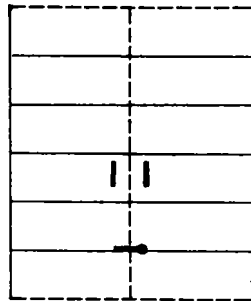


Figure 3.3. Use of basic signs.

The *head* is represented by a straight line drawn between the fourth and fifth stave lines. One end of this head line is fixed to the intersection point of the fourth stave line with the body mid-line, while the other end always touches the fifth stave line. The angle between this head line and the body mid-line specifies the degree of right or left tilt in the head. A shorter straight line drawn perpendicular to the head line represents the chin position. The length of this chin line indicates the degree of turn in the head. The intersection point between the head and chin lines specifies the extent of backward or forward bend in the head as shown in Figure 3.4. Intersection at the mid-point indicates no bend, while intersection above the mid-point represents a backward bend and intersection below mid-point indicate a forward bend. Absence of the head sign indicates no change from the last frame. However, if no head sign is written at the start of a given sequence, the head is assumed to be erect as in the anatomical position shown in Figure 3.1. Furthermore, no head sign need be written until that position changes.

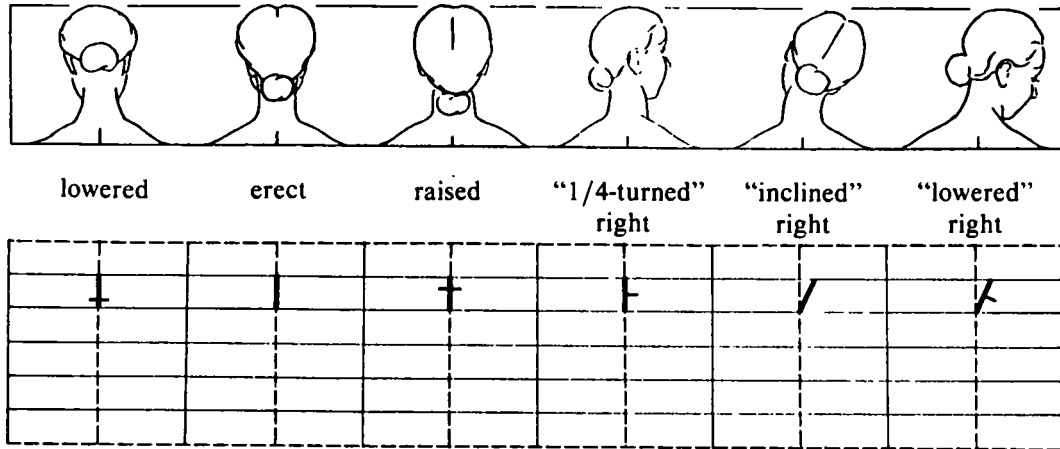


Figure 3.4. Various head positions.

The third stave line represents the waist. Therefore all body bends and twists from the waist up (*upper torso*) are recorded between the third and the fourth stave lines. The basic signs used and their manipulation are identical to those used for the head. Figure 3.5 illustrates the method for recording these main body parts. Hip (*pelvic*) movements are recorded similarly between the second and the third stave lines.

The notation explained so far can provide the record of any static posture and a series of postures can record a movement. But actual movement often consists of very smooth motion. Benesh notation makes use of *movement lines* to record continuous smooth movements. A movement line traces the path of movement in space from the starting position to the final position and summarizes an infinite number of intermediate positions. It is attached to a sign at its final position but not at the starting point, its inception, thus giving direction to the movement as shown in Figure 3.6. If the moving limb travels in a forward or backward curve from its starting point, the movement lines are qualified using the *qualification* signs:

- the forward-most displacement (in front of the coronal plane)
- the back-most displacement (behind the coronal plane)

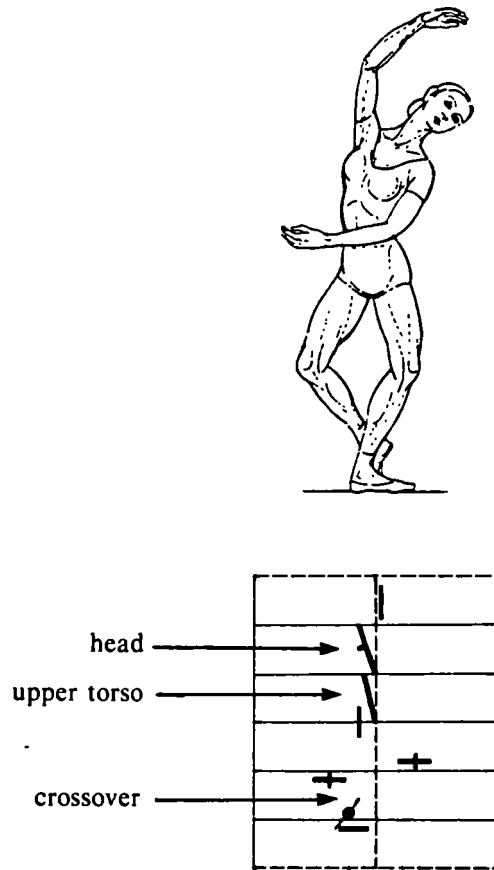


Figure 3.5. Main body signs.

The *forward-most* or *back-most* displacement of a movement is indicated by a short straight line segment intersecting at right angles to the movement line at the point of maximum forward displacement, or a dot on the movement line drawn at the point of maximum backward displacement, respectively. No qualification is necessary for movements within the coronal plane or within a plane parallel to the coronal plane.

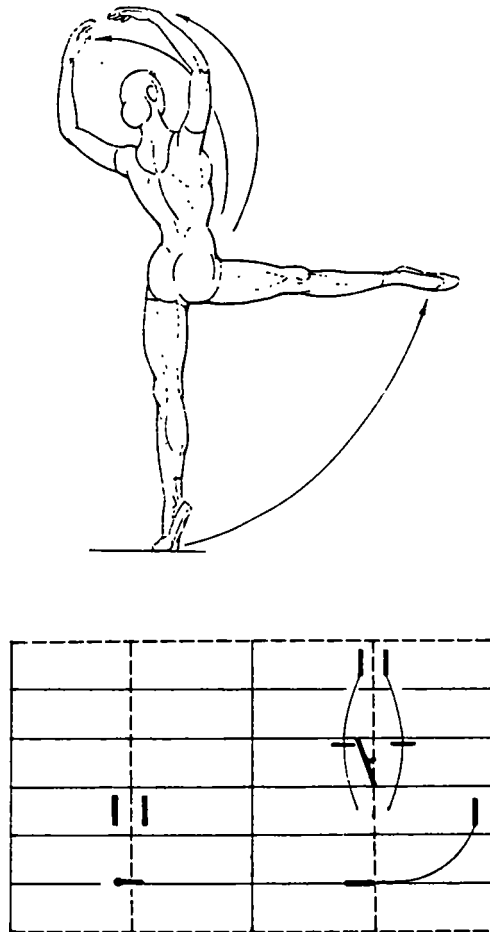


Figure 3.6. Movement lines.

For movements in which the body leaves the ground (*jumps*), the movement path is treated as a special case. The movement line may span several frames. It begins under the *take-off* position, is attached to the *landing position*, and is drawn between the first stave line and the leger line below it. The frames between the take-off and landing positions, if any, describe movements in the air. The recorded movement line is a mirror-like *reflection* of the actual movement path about the first stave line, as shown in Figure 3.7.

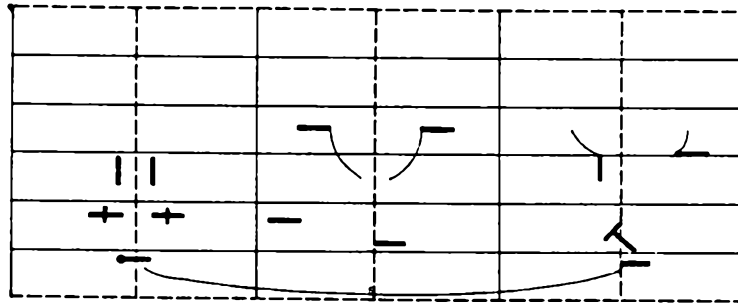
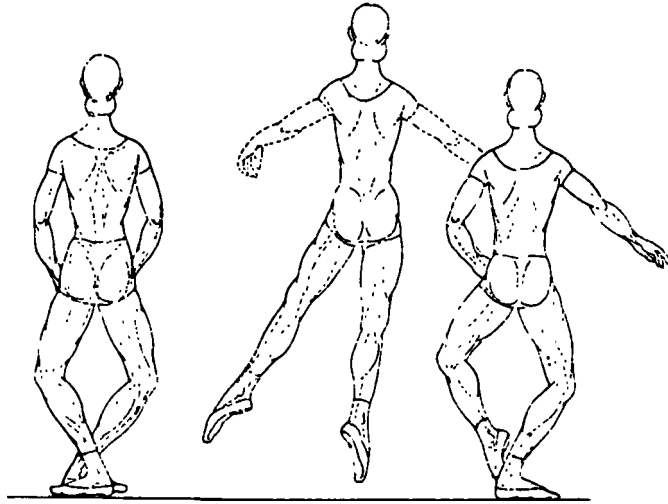










Figure 3.7. Jumps.

Benesh notation uses *direction* signs to state the orientation of the performer relative to the observer. The eight basic direction signs are shown below:

-  facing the audience (*upstage*)
-  facing backward (*downstage*)
-  facing stage right
-  facing stage left
-     facing other directions



Shades of effort, changes of effort and other qualities of movement are recorded using the expression marks of music. In music loudness is indicated by the letters *f* (*forte*) and *p* (*piano*). These letters are used in the Benesh system to indicate the degree of effort in movement. The seven scales of *effort* are:

<i>ppp</i>	completely relaxed
<i>pp</i>	very soft
<i>p</i>	soft
nothing	normal
<i>f</i>	strong
<i>ff</i>	very strong
<i>fff</i>	maximum strength

Changes in effort are shown in the same manner as in music: diverging lines indicate increasing effort while converging lines indicate decreasing effort. Letters at the beginning and end indicate effort at these points as shown in Figure 3.8. The degree of effort is recorded above the stave lines.

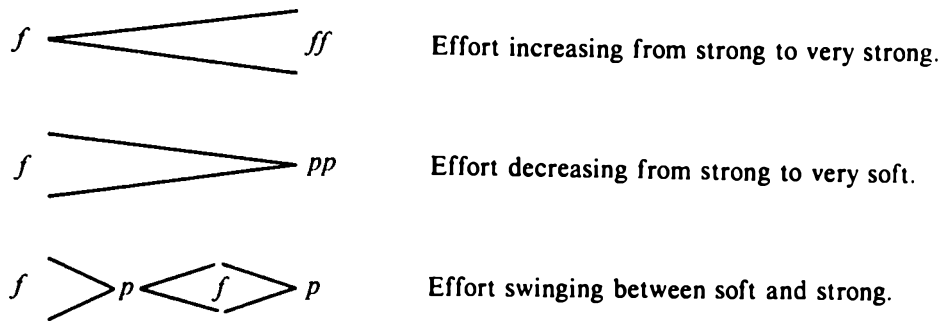


Figure 3.8. Changes in effort.

In past centuries musicians have developed technical terms to describe the character, mood and speed of music. Benesh notation has adopted these terms to describe the prevalent quality of movement [Ryman 82b].







#### TEMPO

<i>Lento</i>	slow
<i>Largo</i>	broad
<i>Adagio</i>	"at ease", slow, drawn out
<i>Grave</i>	heavy, serious
<i>Larghetto</i>	not so broad as <i>Largo</i>
<i>Andante</i>	"going", "walking", medium slow
<i>Moderato</i>	moderate
<i>Allegretto</i>	less fast than <i>Allegro</i>
<i>Andantino</i>	an <i>Andante</i> of small proportions
<i>Allergo</i>	fast
<i>Vivace</i>	lively
<i>Presto</i>	very fast
<i>Vivicissimo</i>	even livelier than <i>Vivace</i>
<i>Prestissimo</i>	even faster than <i>Presto</i>

### MANNER OF PLAYING

<i>Ad Libitum</i>	at liberty
<i>Agitato</i>	agitated
<i>Cantabile</i>	in a singing fashion
<i>Dolce</i>	sweetly
<i>Dolente</i>	sadly, grieving
<i>Sforzando</i>	"forced, accented"
<i>Grandioso</i>	grandly
<i>Legato</i>	smoothly, slurred; end of one note meeting beginning of next
<i>Marcato</i>	in a marked or emphatic manner
<i>Pesante</i>	heavily
<i>Saltando</i>	leaping (strings and springing bow)
<i>Scherzando</i>	jokingly, playfully
<i>Sostenuto</i>	sustained
<i>Staccato</i>	detached, short; each note separate

So far we have considered movement in isolation. To link movement to rhythms and phrases we divide the staff into *bars* consisting of *beats*. In Figure 3.9, the number appearing in the top left corner of the staff corresponds to the musical *time signature*. It indicates the number of beats to a bar. All signs for movement rhythm appear above the staff and indicates whole beats or fractions of beats. They do not indicate the duration of time, but rather the precise moment when a position is reached. The basic rhythm signs are:

	<i>pulse</i>	beat specifies "the count", e.g. one, two, etc.
	<i>te</i>	specifies 1/4 beat after the count
	<i>an</i>	specifies the beat halfway between counts
	<i>ti</i>	specifies 1/4 beat before the count
	<i>dai</i>	specifies the beat 1/3 of the way between counts
	<i>dee</i>	specifies the beat 2/3 of the way between counts

Finally, Appendix B shows a sample Benesh score. The above explanation is only a summary of the basic Benesh notation system. It can be logically extended to capture the movements of eye, hand and fingers, which are very important in dance styles like the *Bharat-Natyam*. For a more complete description of the Benesh notation and its extensions the reader is referred to [Benesh 56, 77; Causley 67; Parker 82; Ryman 82b] and to the *Choreologist*, a technical journal of the Institute of Choreology, London, England.



## 4. THE EDITOR

*"Intelligence ... is the faculty of making artificial objects, especially tools to make tools."*

*- H. Bergson*

The basic unit of information in Benesh Movement Notation is the *frame*. Unlike a line of text, which is made up of symbols (characters) written at discrete points in a one-dimensional space, the Benesh frame records body information by means of symbols in a continuous two-dimensional rectangular space known as the *stave*. Just like lines of text in a document, Benesh frames are ordered sequentially within a score. However, the positioning of symbols within a frame is more complex than placing characters in a line of text, as was explained in Chapter 3.

Considering for the moment a frame to be equivalent to a line of text, the editing requirements for Benesh scores are similar to those for text. For example, three levels of basic operations are necessary for editing a text document: editing lines, editing words within a line and editing characters within a word. Similarly, the editing operations required for Benesh scores are: manipulation of an entire frame or a set of frames, manipulation of symbol positions within a frame and construction of the symbols themselves. There is a one-to-one correspondence between the text and Benesh score entities. For example, a complete score corresponds to an entire text document. High level operations such as *print*, *copy* and *archive*, for Benesh scores are analogous to those for text. The techniques for implementing text editors are well developed and a number of very good editors such as *Vi* [Joy 80], *Emacs* [Finseth 80], *Xerox Star* [Seybold 81a, 81b], *Qed* [Michlin 75], *Fred* [Gardner 80], *Ed* [Kernighan 78a, 78b], *Ex* [Blau 80], and *K* [Pettis 78] are available. The Benesh editor described in this chapter uses many existing techniques by taking advantage of the analogy between editing operations for text documents and for Benesh scores.

The primary objective of the editor is to improve the efficiency of editing Benesh scores and the quality of the final score. A secondary goal is to assist this editing process by providing a pleasant working environment. (The latter of course promotes the former.) This requires a powerful and easily understood command set, appropriate user feedback, and quick system response to commands made by the user. For the purpose of this discussion, the editor's design is divided into three parts. The *data structure manipulator* performs the actual editing, the *display handler* provides user feedback,

and the *command loop* translates user requests into calls to the data structure manipulator and the display handler. Each part of this structure contributes in its own way towards providing quick system response.

#### 4.1 Data Structure Manipulator

Movement can be thought of as a smooth transition between static postures that differ slightly from one another. For this reason any frame of Benesh Movement Notation records only the changes in posture since the last frame, thus reducing the amount of information displayed. This greatly simplifies the process of sequentially reading a score. However, it makes it very difficult to read a static position somewhere in the middle of the score. For editing, it is essential to show the complete body information so that the user can make the necessary changes to it. To do this, one would have to read backwards from the desired frame and gather all body information that is not specified explicitly in the frame. This could require searching backwards to the beginning of the score, a very time consuming task. This problem can be resolved if complete information is stored for each frame but only the changes since the last frame are actually displayed.

As in most text editors, the primary data structure used here is a doubly linked list of frame nodes as shown in Figure 4.1. For the reasons discussed above each node contains complete body information for a frame. Thus the need for a backward search is eliminated. In addition, edit operations like *delete*, *edit*, *move*, and *copy* are simpler to execute because only the part of the linked list specified in the command must be updated. Since each frame is an independent unit, the changes do

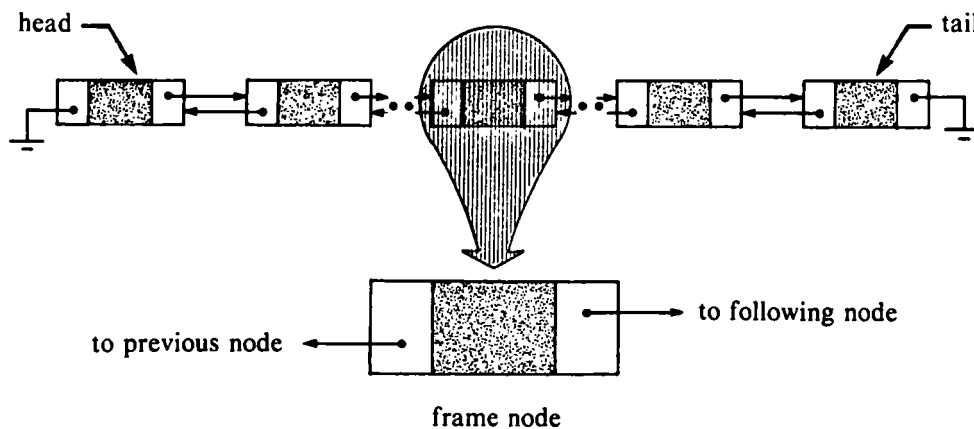


Figure 4.1. Linked list storage structure for Benesh score.

not have to be propagated to subsequent frames in the score. Keeping complete body information for each frame increases the storage overhead, but the system response is much improved. In our editing environment this time/space trade-off seems to be justified.

The basic operations for editing a Benesh score can be categorized according to the entity of information they deal with: frames, symbol positions within a frame, or the symbol themselves. Each of these categories is described below.

#### 4.1.1. Manipulation of Frames

Editing operations on frames relate to positions in the linked list. Thus each frame edit command is mapped into an equivalent operation that manipulates the linked list structure. In performing these operations, the editor uses a temporary storage area known as the *save buffer*. The structure of the save buffer is identical to that of the linked list. There are four basic routines used to manipulate the linked list and the save buffer.

*Add\_Fnode( before, new\_node )*

where *before* is a pointer to a linked list node and *new\_node* is a pointer to a new frame node. This routine inserts the node given by *new\_node* in front of the node specified by *before* in the linked list structure.

*Delete( start, end )*

where *start* and *end* are pointers to the first and last nodes of a sequence of frames in the linked list. This routine deletes the specified section from the linked list structure. Deleted frames are not simply discarded, but saved in the save buffer in case an *undo* or a *Put* command needs to restore them as explained later in this chapter.

*Save( start, end )*

where *start* and *end* are pointers to the first and last nodes in a sequence of frames in the linked list. This routine copies the specified sequence into a save buffer.

*Put( before )*

where *before* is a pointer to a linked list node. This routine inserts a copy of the save buffer immediately in front of the specified node. The contents of the save buffer are left intact for another *Put* command.

All higher level operations on the data structure are performed by calling these basic routines. For example,

*Copy( start, end, before )*

where *start* and *end* are pointers to the first and last nodes of a subsequence in the linked list and *before* is a pointer to a frame node in the linked list. This routine copies the specified sequence of frame nodes into the linked list immediately in front

of the frame node specified by *before*. This operation is equivalent to the following sequence of calls to the basic routines:

*Save( start, end )*  
*Put( before )*

*Move( start, end, before )*

where *start* and *end* are pointers to the first and last nodes of a subsequence in the linked list and *before* is a pointer to a frame node in the linked list. This routine inserts the specified sequence of frames into the linked list immediately in front of the node specified by *before*, and deletes the subsequence from its original position. This operation is equivalent to the following sequence of calls to the basic routines:

*Delete( start, end )*  
*Put( before )*.

Notice that an error occurs if *before* is between *start* and *end*. However, the user interface routines ensure that such a situation does not arise.

#### 4.1.2. Manipulating Symbol Positions within a Frame

Editing operations at this level deal with the position of sub-components (symbols) within a frame. All new frames are first composed in a working area and then added to the linked list using the basic routine *Add\_Fnode()*. To edit an existing frame, the desired frame is first copied into a working frame. When editing has been completed, the working frame replaces the old frame in the linked list. This is accomplished by the following routine:

*Copy\_Fnode( from, to )*

Where *from* and *to* are pointers to frame nodes. This routine copies the entire contents of the frame node given by *from* into the frame node given by *to*.

Another set of routines is used to modify the body parts within the working frame. These routines ensure that Benesh conventions are observed whenever any information within the working frame is modified. A body part position can be deduced from the symbol used and its relative position within the frame. Since the body model used in the Benesh notation has a fixed number of parts, all body information is stored in an array of pointers which reference structures containing the symbol used and its relative position within the frame as shown in Figure 4.2. Each array element refers to a specific body part. The manipulation of these body parts may involve positioning a new symbol within the working frame, changing its position or replacing it with another symbol. The manipulation of individual body part symbols is discussed in Chapter 5, which deals with the user interface.

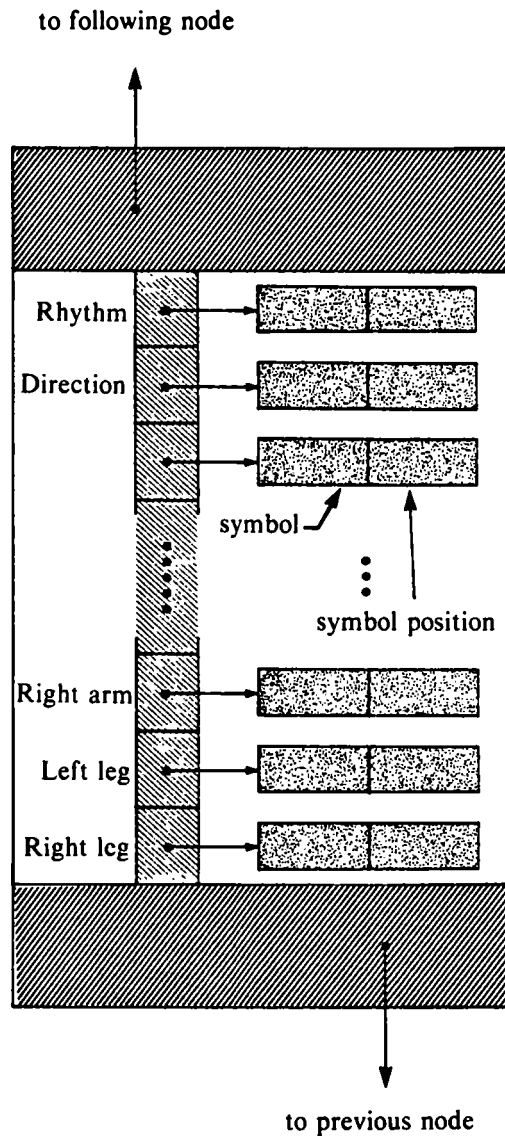


Figure 4.2. Frame node.

#### 4.1.3. Construction of Benesh Symbols

Benesh notation uses a basic pre-defined set of symbols. Others are built from a combination of these basic symbols such as *crossovers*, and still others like main body symbols, are constructed whenever necessary. This last approach is used for symbols which represent multiple, highly variable dimensions, for example, the symbol which captures the degree of tilt, turn and backward or forward bend of the head. In this case it would be very difficult to provide a fixed set of predefined symbols covering all the possible combinations and thus a new symbol is constructed whenever a new head position must be specified. This operation is further described in Chapter 5.



## 4.2. Display Handler

This section of code keeps the system state accurately displayed on the display screen. The goal is to perform this function in such a way as to minimize the amount of *clock time* required in order to make changes. Clock time is the time perceived by the user from the issuance of a command to its completion, including updates to the display. The display routines are also responsible for showing the system state and displaying the appropriate menus, depending on the state.

### 4.2.1. Frame Display

A moving *display window* is superimposed on the linked list structure. At any given time, the editor displays all the frames in the current display window. Other frames are displayed by moving the window over the desired frames in the linked list. To a user, the window displays the score as it would normally be written on a piece of paper, except that the first frame in the window always shows the complete body position for readability, irrespective of the frame immediately preceding it in the score.

The window can display a maximum of sixteen frames arranged as two lines of eight frames each. It consists of an array of pointers to a sequence of consecutive frame nodes in the linked list as shown in Figure 4.3. The following routine is used to move the display window along the linked list:

*Move\_Window( +/- arg )*

This routine moves the display window forward or backwards in the linked list, depending on whether the argument is positive or negative, respectively. The argument *arg* specifies the number of frames the window is to be moved.

The window is generally moved backwards or forwards by eight frames corresponding to a line of score on the display screen.

To add a frame to the display window the user pre-selects the *insert position* within the window. This position could lie between frames or at the beginning or end of the window. The frame immediately in front of the insert position, if any, is copied into the working frame. The *complete* body position is then displayed in the working frame, irrespective of what is visible in the display window. This provides a reference point for continuing the new movement sequence. If no frame exists immediately before the insertion position, such as when inserting at the beginning of the displayed score, a blank working frame is displayed.

When adding a composed frame from the working frame to the display window, the new body position is compared with the body position of the frame immediately in front of the insert position.

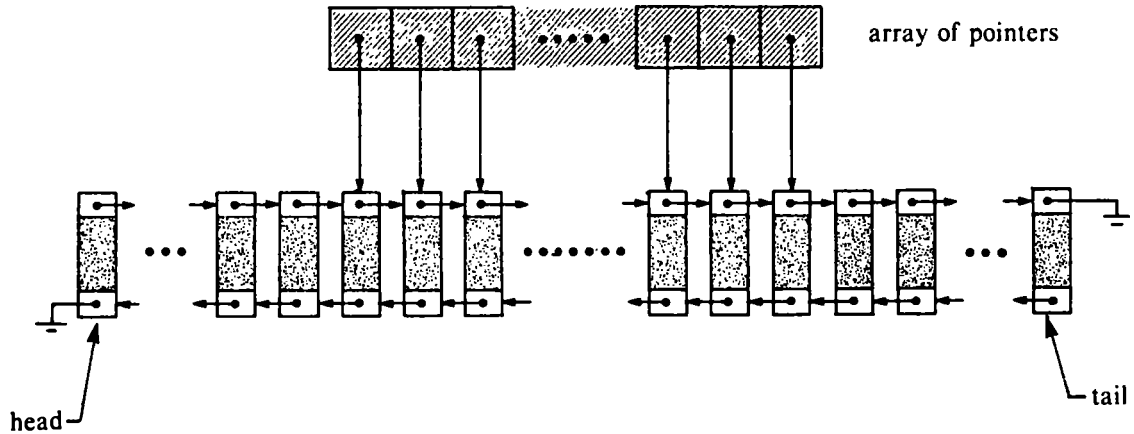


Figure 4.3. Display Window.

Changes in body position are then determined and displayed in the new frame. However, as described earlier, the complete body information is stored in the frame node.

Before adding a new frame to the display window, frames immediately following the insert position are shifted right by one frame position to make room for the new frame. If, while adding new frames, the insert position moves out of the display window, the window is advanced by eight frame nodes within the linked list. If the window were only advanced by a single frame node, the next addition of a new frame would immediately necessitate another shift.

If the display window contains some empty frame positions as a result of the previous operation (such as deletion of a frame sequence), all frames following the empty frame positions, are shifted left to close the gap. If the display window is empty as a result of the previous operation, the window is moved over to the following frame nodes in the linked list, if any, and the new window is displayed.

Benesh Movement Notation uses two different types of frames. A *header* frame defines the time signature and tempo information that applies to the following frames in the score. A *body data* frame defines a body position. The frame following the header frame defines the starting position of the movement sequence. Copying a frame from the display window into the working frame to act as a reference must be done with care. If this frame is a header frame, it cannot act as a reference and thus the working frame is left empty.

#### 4.2.2. Menu Display

The system displays the full range of options available at a given time. Which options are available depends on the system state and the context of the operation being performed. The list of options available is called a *menu*. The editor's command set is organized as a hierarchical collection of menus. The display of the system state and the associated menu is the responsibility of the display handler. The menu format and details of the system state display are described in Chapter 5.

#### 4.3. Command Loop

The command loop implements the control logic of the editor. It is responsible for recognizing user commands, executing them by calling routines for manipulating the data structures, and displaying the results by invoking display routines. The basic loop structure for the editor is:

```
Cycle {  
    Get_Command( x )  
    Execute( x )  
    Update_Display  
} end Cycle
```

The philosophy behind this loop is to put as few restrictions on the user interface as possible. Each user command maps into a executable procedure.

The command syntax is predetermined and implemented as a finite state automaton. Every command causes a transition in the system state and the display of a new menu depending on this transition. Appendix C shows the state transition diagrams for the editor's command language. Since the system displays only the next set of available options, the command semantics are also predetermined and there is very little possibility for error. No parsing is necessary and syntax errors are prevented by the user interface described in Chapter 5. Other errors are possible however, and the procedures for recovering from them are discussed in the next section.

### 4.3.1. Error Recovery

*"To err is human,  
to forgive is design"*  
- Anonymous

As in a text editor, there are two types of errors in the Benesh editor: *internal errors*, caused by problems within the editor, and *external errors*, caused by the user. Internal errors result in an immediate exit to the operating system. They occur because of system bugs or resource restrictions and are identified by error messages in so far as is possible. Ideally the user never sees this type of error. External errors are primarily user errors. Generally the action taken is the display of an error message and a return to command level.

In this editor the basic philosophy about error recovery is to avoid errors and consequently error recovery. By displaying only valid options at a given time most external errors are avoided. However, the user could position a symbol in such a way as to specify an impossible body position. One action would be to allow the user to make such errors, and then display an error message. However, this requires a means of allowing the user to gracefully recover from this error. A much better way would be to make it impossible for users to make such errors. This can be done by dynamically constraining the positioning of symbols to areas that would not result in an impossible body position. Displaying an error message and not allowing the symbol positioning is sufficient if the user tries to place a symbol in an *illegal* area.

Even if the user is restricted to choices that are syntactically and semantically valid, he can select an unintended command by mistake. Hence it must be possible to back out of an undesired state. The user can do this by *aborting* the current action or *undoing* the last action to recover the previous system state.

In addition, the system detects one potential error condition when the user wants to exit from the editor. If the current score has not been archived, the system will display an appropriate warning message asking the user to either archive the score or confirm his intentions to exit without archiving.

### 4.3.2. Abort and Undo

*"O God! O God! that it were possible to undo things done:  
to call back yesterday!"*

*- T. Heywood*

At his discretion, the user can *abort* out of the current system state. *Abort* cancels the current action and backs up the system state to the next higher level in the menu hierarchy. Thus a sequence of *abort* requests will eventually return the editor to its initial state.

The editor also provides a *one step undo* of any command after it has been completed. This allows the user to correct single-level mistakes. Two types of *undo* requests are supported by the editor. The first type of request undoes the previous step in a sequence of actions which comprises an editing operation, while the other undoes the entire last operation that was successfully completed. In both cases *undo* acts like a toggle switch; a second application undoes the last *undo*, thus restoring the original action.

To implement *undo*, the editor keeps track of the current and last system state. To *undo* a single step within an operation, the editor switches between these states. However, undoing the last *complete* operation is more complicated and is further divided into the *undoing of symbols*, and the *undoing of frames*, corresponding to the different levels of editing: editing symbols within a frame and editing frames within a score. When modifying a symbol, the old symbol is not deleted, but saved in an undo structure along with its position and all other relevant information. To execute an *undo*, the editor simply replaces the current symbol with the saved symbol from the undo structure. Adding a new symbol is considered to be equivalent to modifying a non-existing symbol.

In order to *undo* frame manipulations, the editor maintains a *save buffer* where deleted frames are placed. Frame move and copy operations also use this buffer. Thus there is a mechanism by which the user can recover frames which were accidentally deleted. In addition, some internal variables must be saved to restore the finished score display. For example, in the move frame operation the saved variables are:

- a pointer to the first frame in the sequence and its display position within the window,
- a pointer to the last frame in the sequence and its display position within the window, and
- the new position in the display window.

The display window positions and pointers to the frame nodes displayed at those positions are necessary to restore the finished score display just in case the display window was moved during the last operation.

This chapter has described the functions and mechanisms supplied by the editor for creating, manipulating and storing Benesh scores. To ensure that these functions are used correctly and in a consistent manner, the user does not invoke these mechanisms directly but is instead presented with a higher level abstraction as shown in Figure 1.1. This abstraction, the *user interface*, is described in detail in the following chapter.

## 5. THE USER INTERFACE

*"It is easy to make things hard.  
It is hard to make things easy"*

*- A. Chapanis*

The most important single consideration in designing any computer system, hardware or software, is the design of the interface between computer and user. It is the most visible aspect and the only channel of communication between the system and the choreologist. The success of a system is entirely dependent on the success of its user interface. However, user interfaces are the most difficult and the least understood part of interactive systems. The design process is very iterative and thus very time consuming. For example, Xerox spent over 20 man-years developing the user interface for their Star work-station and experimenting with alternate designs [Smith 82]; the result is a finely tuned user interface.

For a successful user interface, the design must include a *task analysis* phase. The designer must analyze current tasks performed by the user, prior to introducing the computer system. Foley points out that "the first step is to understand the problem area and the prospective users" [Foley 82a]. Hansen advises, "know your user" [Hansen 71]. Task analysis is an essential step in gathering information necessary for making decisions throughout the design process. It involves interviewing the prospective users and studying their working environment to determine which tasks can best be aided by a computer. This study results in a set of design objectives, constraints and functional requirements of the capabilities to be made available through the user interface. This first analysis can often provide an insight into how these capabilities should be presented to the user. It also identifies the group of users for which the system is being designed, their performance goals and the methods they use to achieve those goals. A description of the current working environment with a breakdown of its information entities and methods employed offers a starting point for the design process [Hornbuckle 67]. The idea behind this phase of design is to develop a *new environment* in which the user can work to accomplish the same goals as before, but using a different set of objects and employing new methods. The new task environment often adds new goals which were not feasible in the original system.

Task analysis itself requires a considerable amount of skill and experience. However, it greatly simplifies the remaining steps in the design process. When designing a user interface, two primary issues of concern are the selection of:

- an *interaction language*, the medium through which the user may express commands to the system, and
- a *display representation*, which shows the system state and the task status in response to user commands.

The first is expressed with actions applied to the input devices, while the latter is expressed through the output devices.

*Prototyping* is a crucial step in the design process. It involves implementing a new concept and testing it with intended users. If the concept doesn't work, other ideas are tried until a suitable solution is found. Prototyping presents two problems. First, the user generally has no idea of the facilities that new technology can provide to aid in his task, so the designer must constantly present alternatives from which the user may choose. Secondly, it may be difficult to decide whether a concept works or not. The user often cannot help in this decision because of his lack of knowledge of the design process. The designer has to develop methods for aiding this decision process.

The interactive techniques used by the Benesh editor have evolved through just such a task analysis and prototyping process. A choreologist was closely observed at work under rehearsal conditions and later interviewed. The results of this study formed the problem description described in Chapter 2. Close observation of user behaviour and a trace of interactions during editing sessions helped in isolating problem areas and devising alternate techniques. The resulting design adheres to a small set of principles that have been adopted to make the editing environment familiar and friendly to the user, to simplify the man-machine interaction, and to unify all editing functions so that any experience gained in one situation can be easily applied to similar situations. This chapter describes these principles and illustrates each with examples from the Benesh editor.

## **5.1. Hardware and Software**

Before describing the user interface of the Benesh editor, several essential characteristics of the hardware and software should be pointed out. Without these it would not have been possible to design an interface such as the present one.



The editor uses an Ikonas RDS 3000 *frame buffer* attached to a *colour display monitor* and a Summagraphics Bit Pad *tablet* with a *puck*. The frame buffer consists of a large block of *memory* (512x512x32 bits), a *video generation system* that displays the contents of the entire memory 30 times a second, an *interface* to the PDP 11/45 host minicomputer, and a dedicated *bit-slice microprocessor* directly coupled to the frame buffer. A *pixel* is a 1x1x32 bit section of the frame buffer representing a dot in the RGB (red, green and blue) colour space of the display monitor.

The Ikonas video generation system, as shown in Figure 5.1, consists of a *control module*, a *crossbar switch*, and a set of *colour lookup tables*. The control module reads the appropriate bits for each pixel from the frame buffer memory for input into the crossbar switch. The crossbar switch allows reconfiguration of the 32 bits into any desired output format. A typical configuration is eight bits of red, eight bits of green, eight bits of blue and eight bits for overlay. A colour lookup table consists of a set of high speed registers that perform the mapping from pixel values stored in the frame buffer to the digital colour levels sent to the D/A converters. As each pixel is accessed during the display cycle, the various bit planes are read to form eight bit colour numbers, one each for red, green and blue (RGB). These numbers act as *indices* into the red, green, and blue colour maps where the actual digital intensities to be displayed are generated [Booth 82; McKay 82b]. The editor makes extensive use of the colour lookup tables and the bit-slice microprocessor as explained below. Despite the high depth resolution available on the Ikonas frame buffer, only eight of the thirty-two bit planes are actually used in the editor. This will simplify the anticipated re-implementation of the editor on a lower cost dedicated work-station.

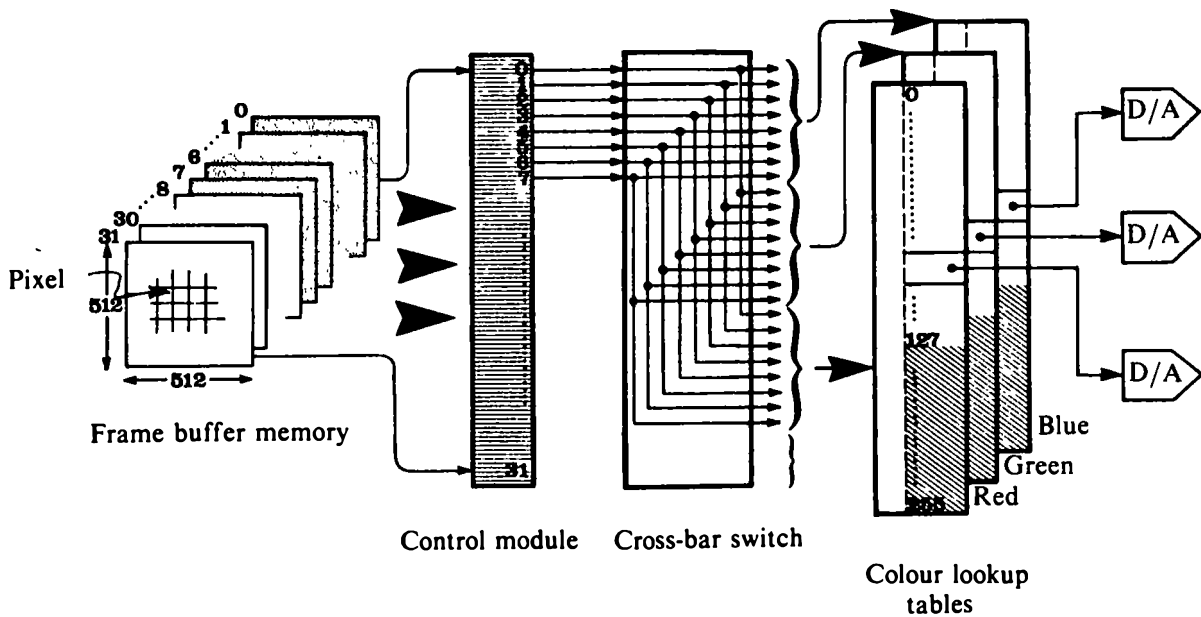


Figure 5.1. Ikonas video chain.

For interaction purposes the eight bit planes are divided into two sets called the *foreground* and *background* as shown in Figure 5.2. The background planes contain static display information, such as the static menus explained later. Figure 5.3 shows the layout of information in the background planes. Each item of static information is written into the frame buffer memory using a different eight-bit colour value. The item can then be made visible or invisible by manipulating colour lookup table entries. To display an item written in the background planes, the corresponding colour lookup table entry is set to a colour different from the background colour. Similarly, to turn off the display, the corresponding colour entry is set to the background colour. This technique is commonly known as *colour table animation*, and several variations of it are described in [Shoup 79; Booth 82]. Since the background consists of seven bit planes, corresponding to entries 0 through 127 in the colour lookup tables, a maximum of 128 different items can be written and displayed independently in the background planes.

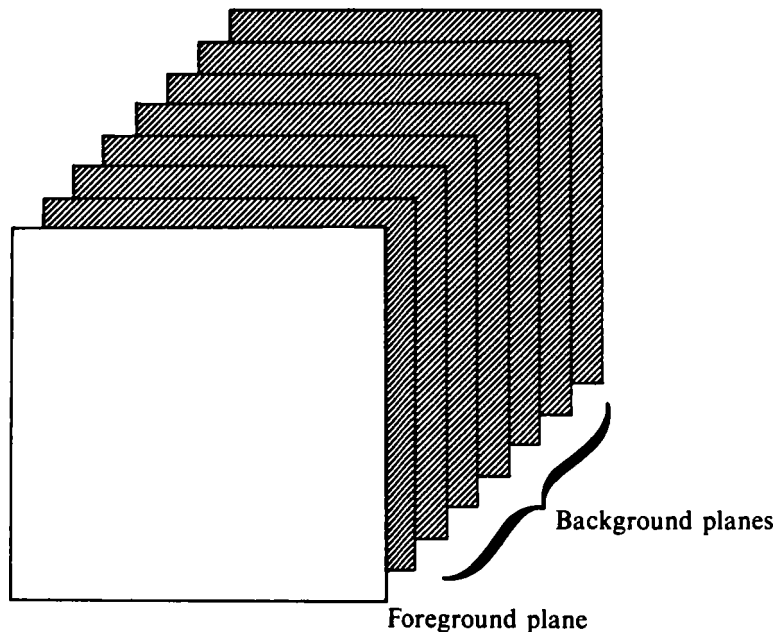


Figure 5.2. Foreground and background planes.

The foreground plane contains dynamic information, such as dynamic menus. A support package developed by Paul Breslin is used to display all foreground information [Breslin 82]. This system is organized around a segmented display file which is interpreted by the high speed bit-slice microprocessor. It provides facilities to *create*, *delete*, *move*, or *pick* segments and to make them visible or invisible.

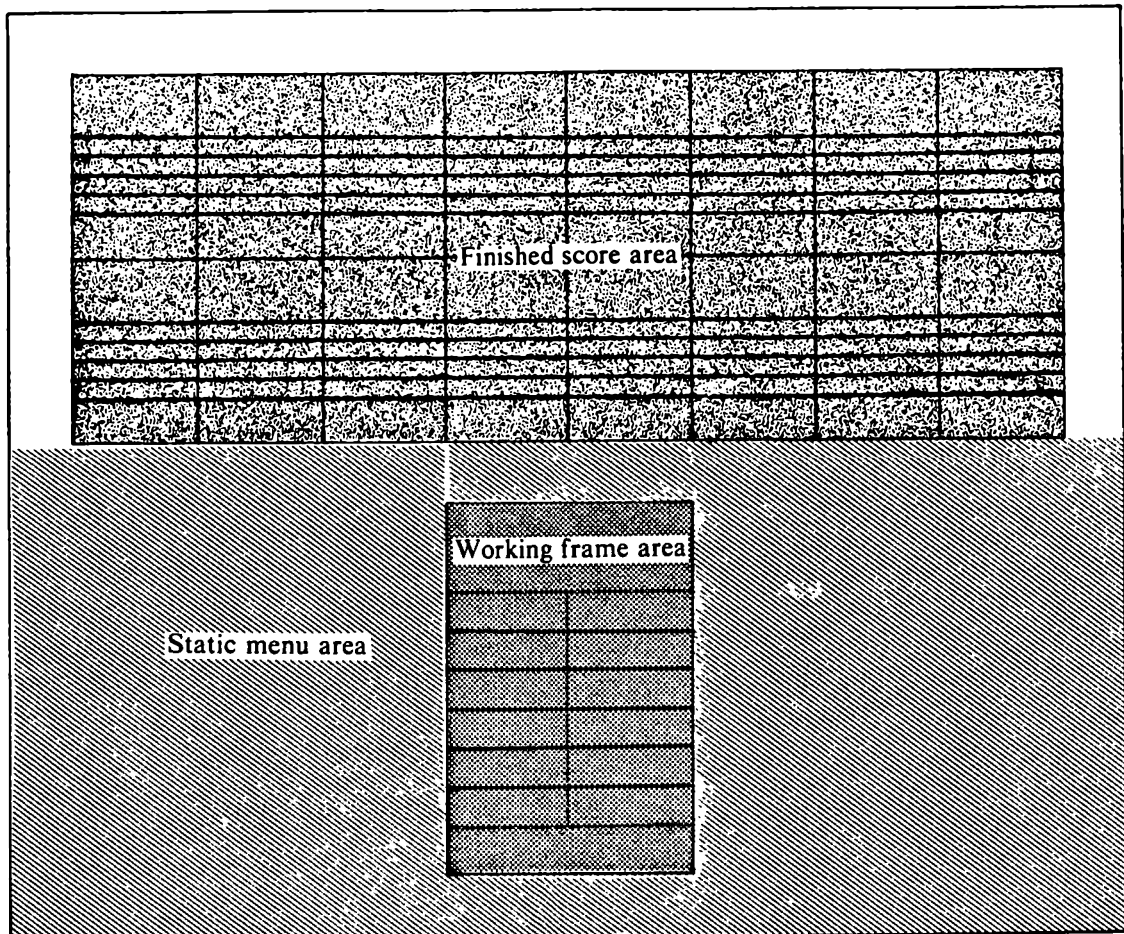


Figure 5.3. Display layout of information in the background plane.

The editor runs with the frame buffer *write-mask* set to manipulate only the foreground plane and with the *auto-clear bit* set to clear the foreground plane after every frame display cycle. This combination of clearing and re-writing segments provides the effect of dragging a symbol if its display position is changed before the next display cycle. Since all of this is done in the microprocessor, the redisplay appears to be instantaneous. As long as the number of segments to be displayed is kept small this results in very smooth dragging.

All code for the editor is written in the C programming language and runs on a PDP 11/45 minicomputer under the Unix operating system.

## 5.2. Interaction Language

*"The best part of human language properly so called, is derived from reflection on the acts of the mind itself"*  
- S.T. Coleridge

The user communicates with the editor through the *graphics tablet* and a *puck*. As the user moves the puck across the tablet surface, its horizontal and vertical position is determined and a *tracker* symbol is displayed at an equivalent position on the display screen as shown in Figure 5.4.

The editor is based on a *menu-driven* scheme. A list of options (the *current menu*) is displayed, from which the user chooses an action by moving the puck until the tracking symbol is aligned with the desired menu option on the display screen. Pressing one of the puck buttons selects the action. The displayed items can specify actions or they can request a different menu. As explained in [Newman 79], the menu-based scheme offers several advantages. A menu displays the full range of options available to the user. This avoids the problem of erroneous commands by preventing the user from selecting options outside this range. This also helps the user to remember commands which he does not use very frequently. In addition, it is very flexible: a menu label can be easily changed to suit the user's terminology, whereas relabeling a set of function keys or adding a new key can be very time-consuming and expensive for the designer.

The prime objective in designing a suitable interaction language is to simplify communication between the editor and its users. The following sections explain the principles used in designing such a language.

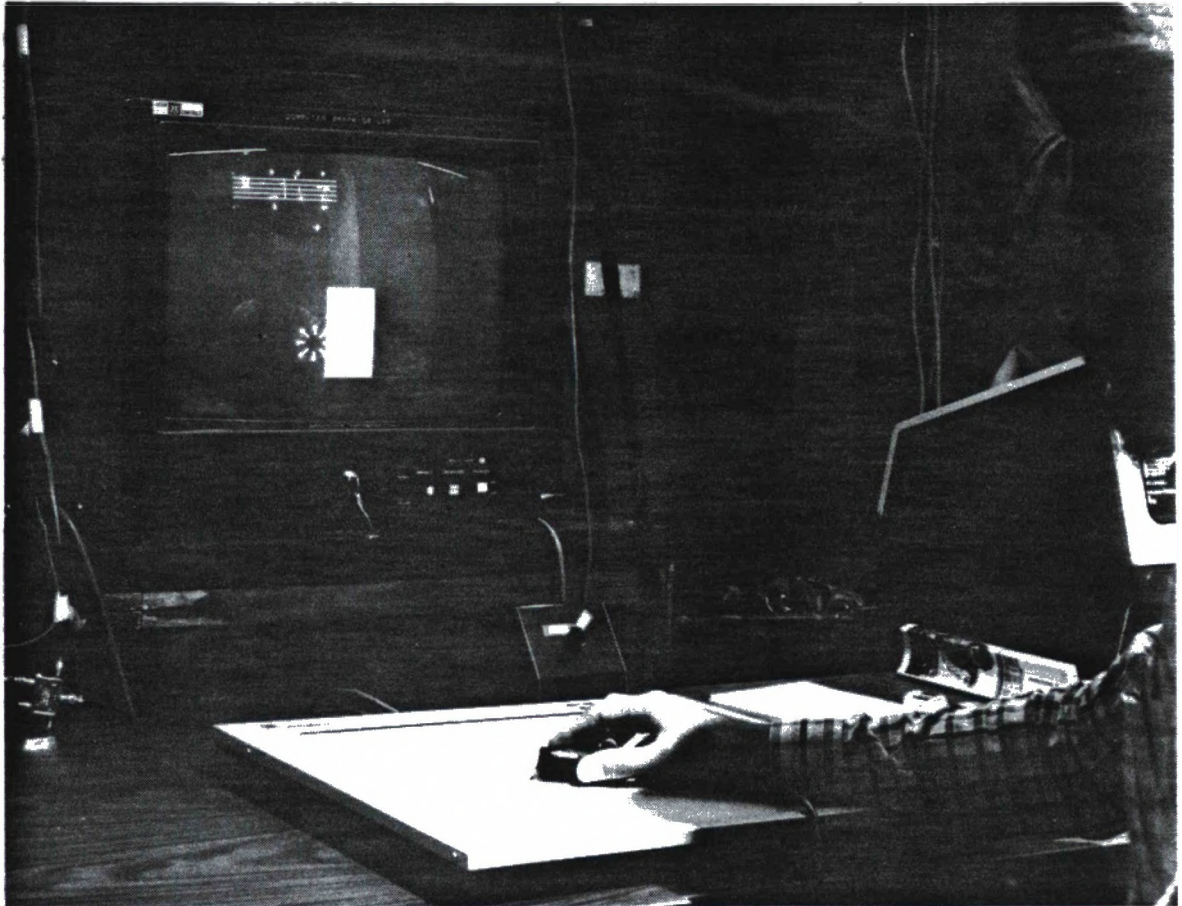


Figure 5.4. Display screen with tablet and puck.  
(photo by Photographic Services, University of Waterloo).

### 5.2.1. The Conceptual Model

*"Conceptuality requires generalization of patterns gleaned from special-case experiences"*

*- R.B. Fuller*

A user's *conceptual model* is the set of ideas which are used to explain the behaviour of the system. It is the picture developed in the mind of the user which enables him to understand and interact with the system. Developing a suitable conceptual model is a very important step in the design of an interaction language; the approach adopted will strongly influence the function of the system.

The designer can choose to adopt analogies familiar to the user and mimic the existing task environment, or he can introduce entirely new functions requiring new approaches. Each approach has its advantages and disadvantages. A conceptual model already familiar to the user will certainly help in adapting to the new system. It will also help in user acceptance of the new system. However, as the user gains experience with the system, he may require short cuts in specifying commands to perform particular operations, making the mimicking approach inefficient.

On the other hand, a new approach provides the means for major improvements in the methodologies for performing existing tasks and prevents the user from misconceptions carried over from the previous method. However, this increase in efficiency has to be balanced against the training and adaptation costs for switching to new methods. A typical strategy is to first computerize the existing methods and then switch to better and more efficient methods as the user becomes familiar and gains experience and confidence with the system [Foley 82a, 82b].

Habitually people prefer the traditional ways of doing things. These methods have evolved from years of experience and most initial difficulties have been ironed out. People have already been trained to use these methods. As a result they feel confident and comfortable using them. It is very hard to change one's way of thinking. In fact, professionals resent being told that their traditional methods are inefficient. This is not meant to imply that new approaches which take advantage of evolving technology should not be introduced, but rather that care must be exercised in introducing new methods.

The conceptual model used in the Benesh editor is identical to that of the Benesh Movement Notation. A few new approaches have been added to aid in the editing process. The use of a five line musical stave to record movements in Benesh Movement Notation represents a human matrix making the notation more visual. This aspect of Benesh notation is maintained and further enhanced

in the editor by using a body figure menu to manipulate different body parts. The finished score is displayed exactly as a choreologist would normally write it on a piece of paper. The terminology used for explaining the editor's functions has been adopted from Benesh notation.

A new approach has been introduced for creating the Benesh score. Instead of asking the user to explicitly draw the Benesh symbols, the editor presents a set of neatly drawn symbols. The user need only select a symbol and position it at the desired location. The rules and conventions for positioning the symbols are those of the Benesh notation and are strictly observed by the editor. All frames are composed or modified in a working frame which is displayed at twice the size of a normal frame in the score. This allows for more accurate positioning and manipulation of symbols. The basic idea is to enlarge any frame area that is too small for accurate manipulation or positioning of a symbol. These approaches have been added to aid in the existing tasks and not merely to introduce new tasks.

### **5.2.2. Recognition versus Recall**

Learning to use any system requires memorization of information. An important consideration in designing a user interface is *the nature of the dialogue* between the system and its user. The aim is to minimize the amount of information a user must memorize in order to initiate and carry out a dialogue with the system.

There are two approaches to the design of man-machine dialogues [Martin 73]. One is the *system-initiated dialogue*, in which the system displays everything relevant to a task and then prompts the user for his next action. The system guides the user. All "conversational" and menu oriented systems belong to this category. The other approach is the *user-initiated dialogue*, in which the user requests an action by issuing the necessary commands. Here the user needs to remember all the available commands and their order of execution to perform a particular task.

During conscious thought, the brain utilizes several levels of memory, the most important being the *short-term memory*. Many studies have analyzed the short-term memory and its role in thinking. The following conclusions stand out:

- conscious thought deals with concepts in the short-term memory [Arnheim 71],  
and
- the capacity of the short-term memory is limited [Miller 56].

When all relevant information is visible, the display relieves the load on the short-term memory by acting like a "visual cache". Thinking becomes easier and more productive. A well designed dialogue can actually improve the quality of user thinking [Smith 82]. Thus a system-initiated dialogue poses very little burden on the user's memory. The conversational nature of this dialogue implies a truly interactive form of communication. It enables the system to give instantaneous feedback to the user in response to his previous action. This further enables the system to provide positive or negative feedback, in addition to prompting the user for the next step. However, there is a drawback: the dialogue steps are predetermined and follow a fixed sequence.

User-initiated dialogues, on the other hand, are generally easier to implement and more efficient in executing a particular task once the user has gained some experience with the system. They are usually more flexible and the user can use short cuts to execute functions to perform a particular task. The ordering of dialogue steps is not fixed and thus more efficient for an experienced user. But the opposite is true for a novice or casual user. Thus a system-initiated dialogue is limited due to the specificity of the actions involved, while a user-initiated dialogue is less limited but presents an additional burden on the user's memory.

In designing the man-machine dialogue for the editor, the user's problem space has been broken into tasks and distributed among the different forms of dialogue. Commands like *abort* and *undo*, that can be invoked at any time, are implemented through user-initiated dialogue. The commands in this group have been restricted to a very small number and are structured to fit the user's normal way of thinking and working, thus reducing the added burden on the user's memory resulting from this type of dialogue. The remaining commands are implemented through system-initiated dialogue. This allows for an appropriate grouping of commands within the system-initiated structures, as explained in the next section.

### 5.2.3. The Command Set

The editor has a few commands that can be used throughout the system. These commands, *abort*, *undo*, and *help* (not yet implemented) are user-initiated and can be invoked at any time during an editing session by pressing the puck buttons as shown in Figure 5.5. The assignment of these commands to the puck buttons is always displayed on the screen as an aid to the novice or occasional user.

Buttons allow a very convenient and efficient way of specifying functions, but they distract the user, causing a shift of attention from the display to the buttons. With experience a user can learn the button locations and activate them without looking, just as a touch typist no longer looks at the



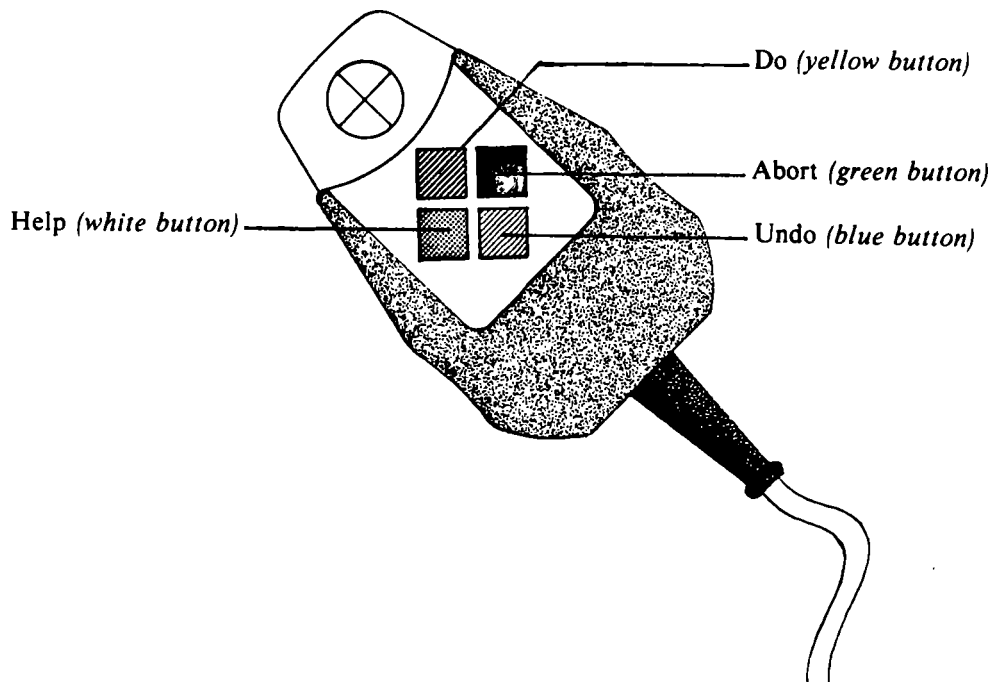


Figure 5.5. Tablet puck

typewriter keys but instead concentrates on the manuscript being typed. This is especially true if there are only a few button positions to remember. However, specifying commands through buttons has other problems. Most function keyboards have no means of labeling buttons under program control. Although it is possible to show a brief summary of the use of each button on a portion of the display screen, the user has to logically connect function assignments to the buttons and so his attention is again diverted. Therefore, we elected to use a minimum number of buttons (i.e. a four button puck) and assign them to the most important functions in the command set. The remaining command set is specified through simulated buttons as menu items on the display screen. Which command is assigned to a button depends on the importance of the function and the frequency with which it is executed during a typical session. Providing a limited number of buttons that are always available within close proximity of the user's current hand position allows the use of tactile memory to assist in locating them.

A menu is commonly used to simulate buttons [Newman 79]. It displays the full range of available functions on the screen. Each item in the menu corresponds to a logical button and is labeled with either a character string or an icon. These labels can be easily changed if necessary. Selecting a menu item by pointing is equivalent to pressing a physical button. Thus menus are suitable for invoking *commands* and also for selecting from a choice of *operands*.

The editor's command set has been divided and grouped into a hierarchy of menus with operand menus at the bottom of the hierarchy. The appropriate menu is displayed on the screen, depending on the system state and the functional semantics at a given time. In designing such a menu hierarchy, the issues of concern are:

- the menu structure,
- the spatial resolution and visual discrimination of items within a menu,
- text versus iconic labels for menu items, and
- dynamic versus static menus.

#### **5.2.3.1. The Menu Structure**

Analyzing the functional requirements in the user's conceptual model, a list of actions can be made for providing these functions. This list forms the vocabulary of the system's interaction language. The list of actions for the editor was divided into functional groups:

- manipulating frames,
- editing frames,
- selecting frames, and
- positioning and manipulating symbols.

#### **Manipulating Frames**

The functional requirements for manipulating frames include adding, editing, deleting, moving and copying frames; selectively displaying sections of the composed score; archiving the composed score and dearchiving it at a later date for further editing. Commands for these functions are grouped together and presented as options in a menu.

A menu can have a very large number of options. The previous discussions in the section on recognition versus recall argued that short-term memory is the most important level utilized in conscious thought. Miller's experiments [Miller 56] show that the capacity of our short-term memory is limited to "the magical number seven plus or minus two" for absolute judgement in unidimensional stimuli. He explains that this capacity can be increased by increasing the dimensions of the stimuli. However, the increase in capacity is not linear. Instead it asymptotically approaches a fixed limit as

the dimensionality of the stimulus is increased [Miller 56]. One way to increase this capacity is by grouping [Bourne 79]. Telephone numbers are divided into groups to help in remembering them. Commands for manipulating frames have been sub-divided into five groups according to their functionality, each group containing a maximum of five commands. Figure 5.6 illustrates this grouping within the *root menu*, the menu for manipulating frames.

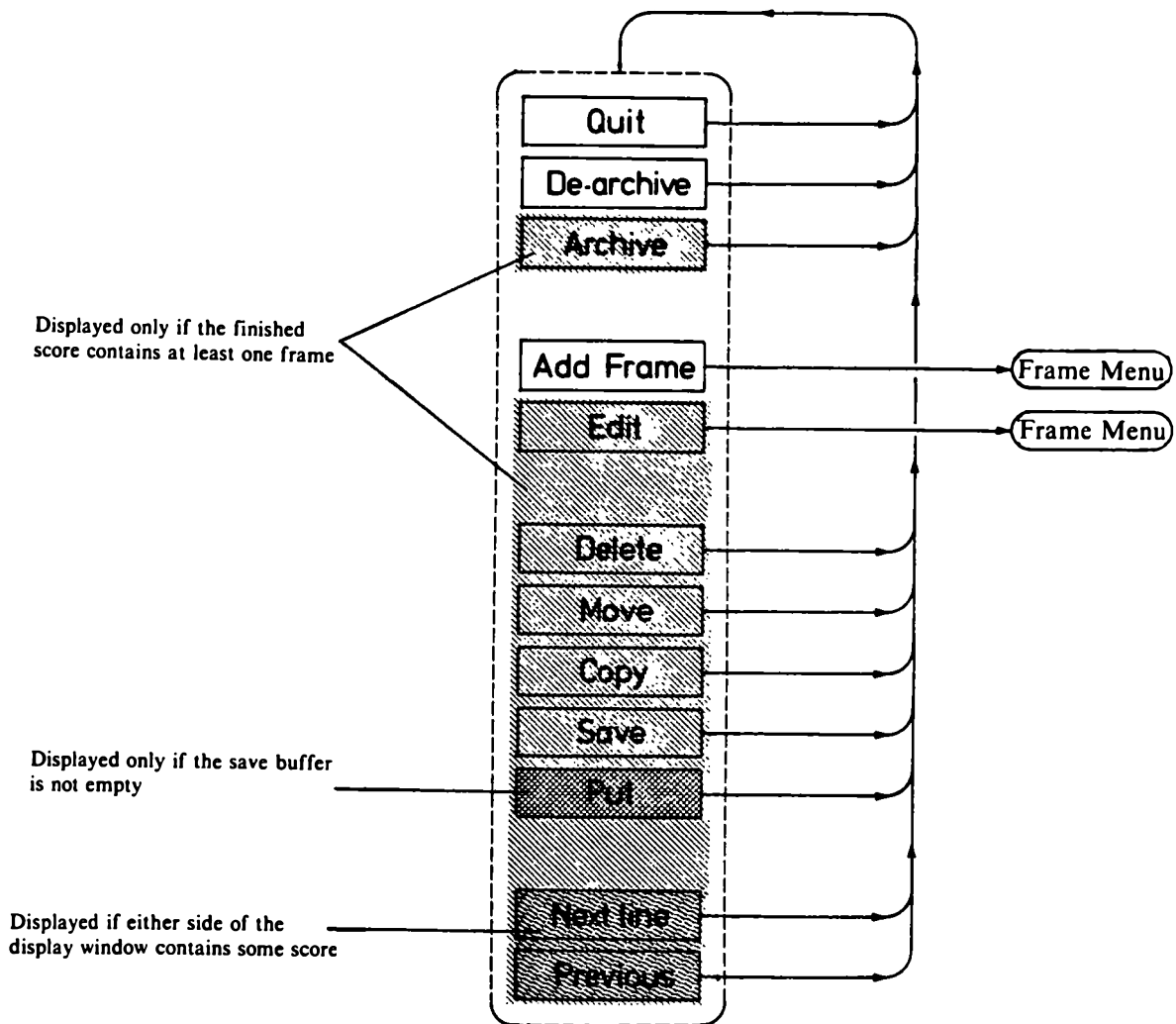


Figure 5.6. Root menu.

### Editing a Frame

Since all frames are composed and edited in the working frame, the functional requirements at this level are specific to the working frame. The functions required for editing a frame can be grouped into two categories: defining a new symbol or modifying an existing symbol, and manipulating the position of these symbols, including the complete erasure of a symbol. Commands in the first set can be further divided according to the type of Benesh frame they refer to, as shown in Figure 5.7. The complete set is always displayed when editing a frame. However, the first selection of an option referring to a specific frame type defines the type for the working frame. Henceforth, the menu displays only options referring to the working frame type and others are eliminated from the display. This menu (the *frame menu*) is invoked by selecting the add or edit frame operation on the menu for manipulating frames shown in Figure 5.6.

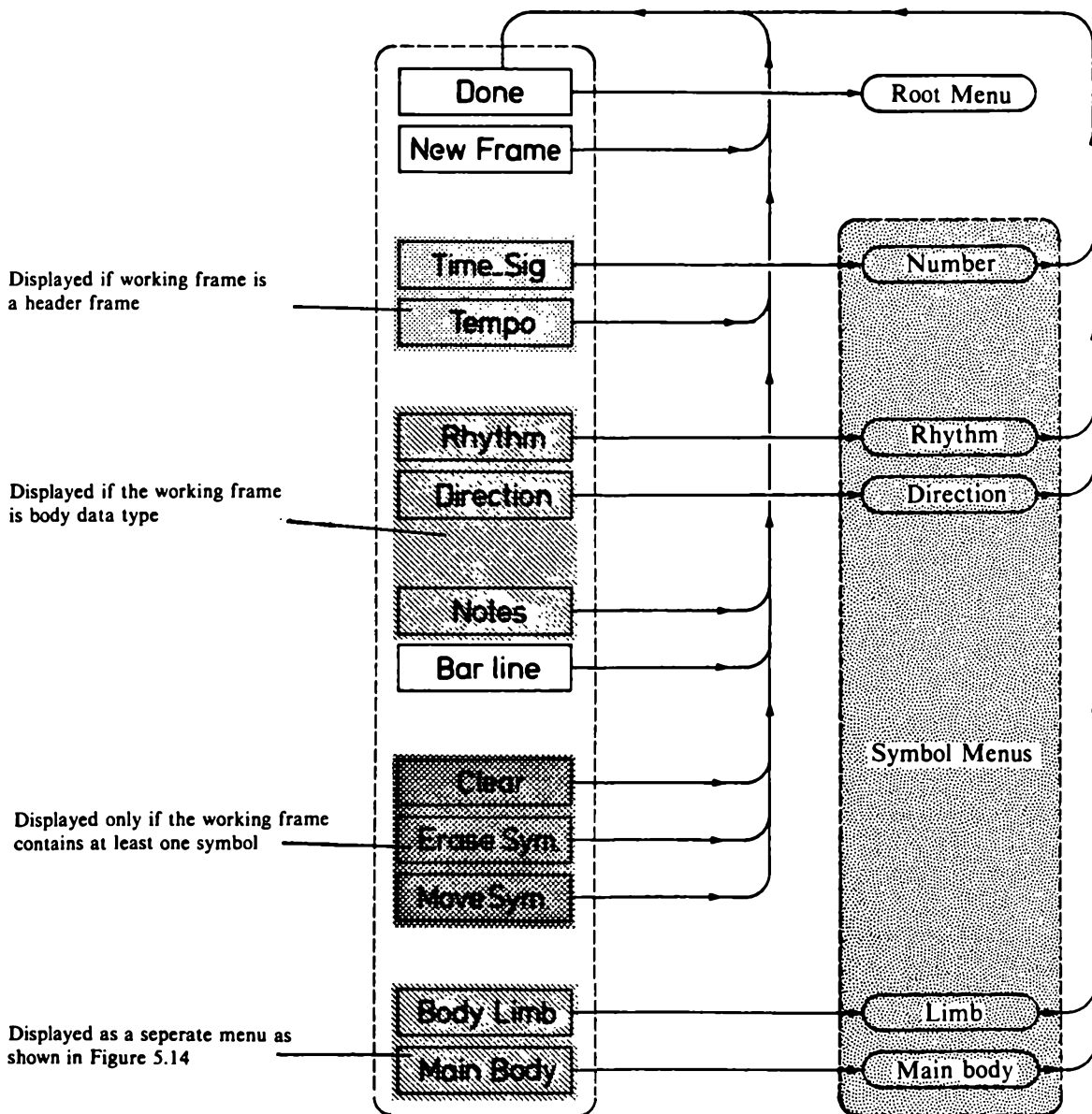


Figure 5.7. Frame menu.

### Selection of Frames

Selection is an integral part of all frame operations such as *delete*, *move*, and *copy*. This section describes the interaction techniques employed in selecting a single frame or a sequence of frames.

To select a single frame the user moves the tracking cursor to the display window. As the cursor moves over a displayed frame in this window, the frame is highlighted by changing its background colour. Once the cursor moves out of the frame the highlight is turned off, as shown in Figure 5.8. Pressing the *do* button on the puck selects the highlighted frame. If no highlighted frame exists when the puck button is pressed, no selection is made.

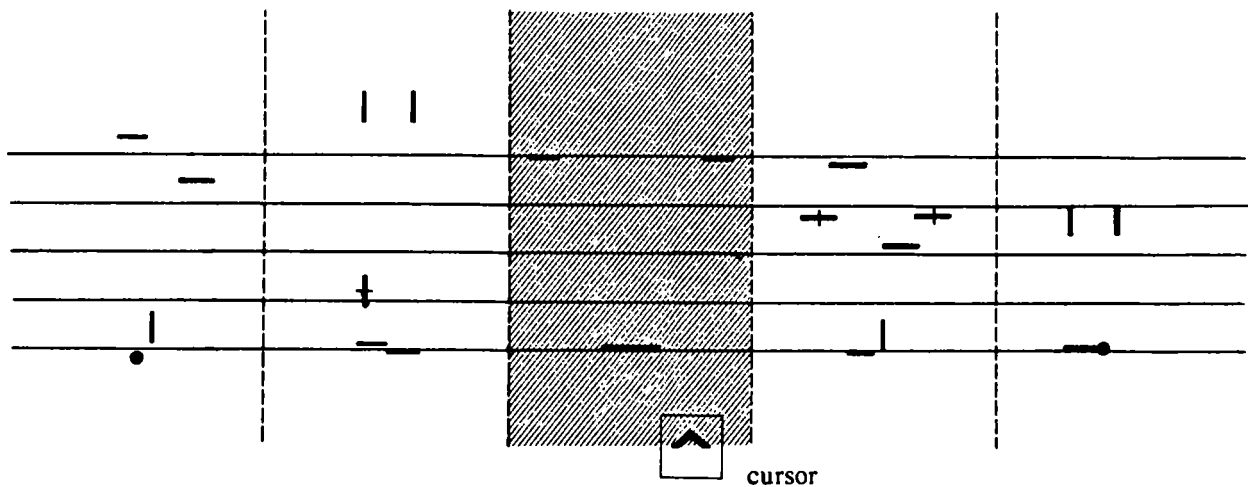


Figure 5.8. Selecting an edit frame.

Adding frames to the display window requires selection of an insertion position for the new frames. To select this position the user moves the tracking cursor over the display window. As the cursor moves across the frames in the display window, an arrow indicating the insertion position is displayed below the frames. The arrow always appears between two frame positions and remains displayed as long as the cursor is positioned in the region between the horizontal mid-points of the previous frame and the next frame, as shown in Figure 5.9.

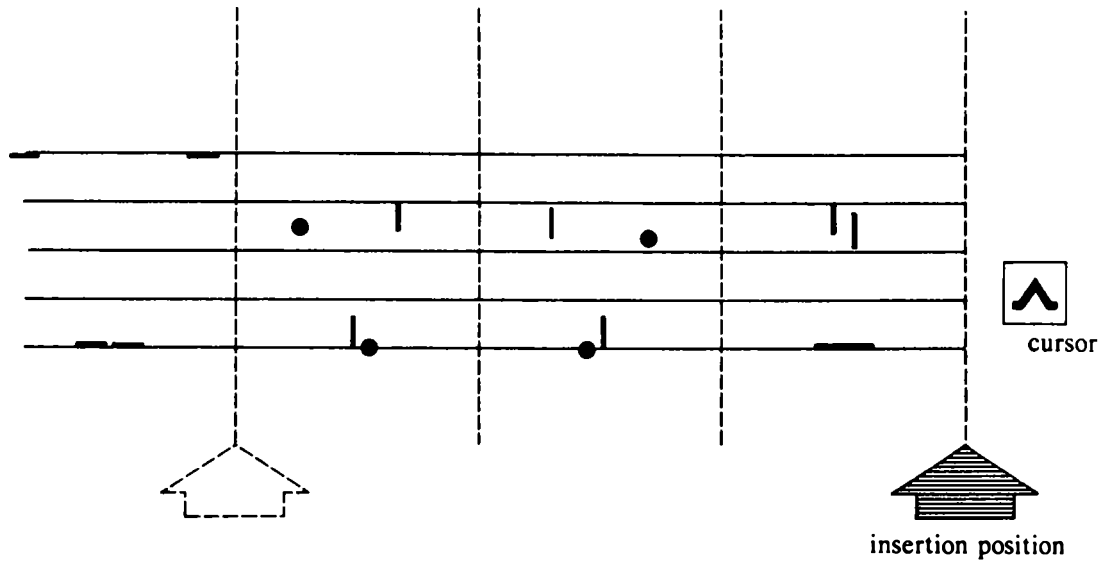


Figure 5.9. Selecting an insertion position.

To select a sequence of frames the user first chooses the beginning frame of the desired sequence, in the same manner as when selecting a frame for editing. Having selected the beginning frame for the desired sequence, all frames between this *anchored frame* and the frame currently under the cursor are highlighted, as shown in Figure 5.10. Pressing the *do* button then selects the highlighted frame sequence. There is no implied ordering on the selection of a sequence; the anchor can be either the first or the last frame in the desired sequence. At any time during the selection, the user can reselect the anchor frame by moving the cursor onto the anchored frame and pressing the *do* button. However, re-selection is only invoked if the user moves to a frame other than the anchored frame before the second *do*. Thus pressing the *do* button twice without moving to any other frame selects a frame sequence consisting of the single highlighted frame.

Moving or copying frames requires an insertion position to which the selected frame sequence is to be moved or copied, respectively. This position is selected in the same manner as when specifying the insertion position for adding new frames to the display window.

### Positioning and Manipulating Symbols

Selecting an item from the frame menu results in the display of the appropriate symbol menu showing the full range of available options for positioning the corresponding frame information. On selection, the symbol is placed in its final position within the working frame, automatically whenever this is known to the system. The time signature and rhythm symbols are examples of symbols which

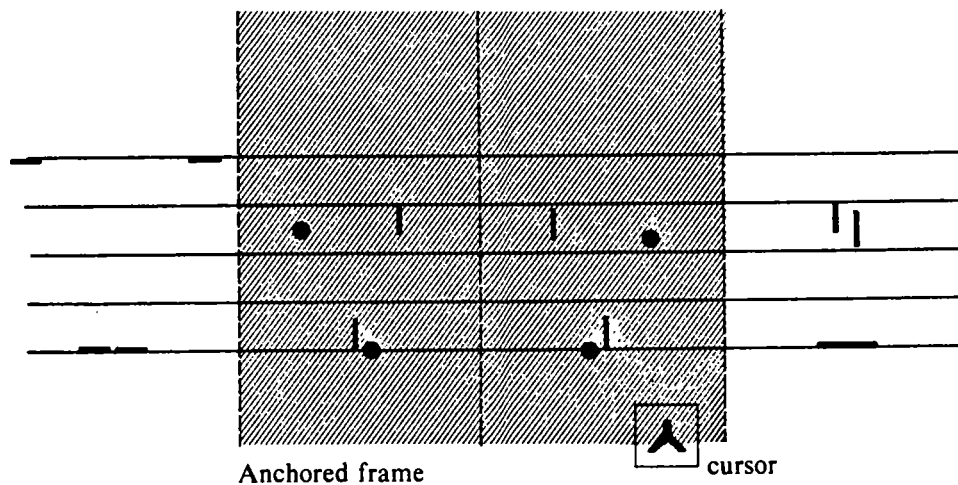


Figure 5.10. Selecting a frame sequence.

always appear in pre-defined positions. On the other hand, if the final display position is not known, the iconic tracker changes its shape to that of the selected symbol and is then dragged to the desired position. The editor observes all the Benesh conventions for positioning symbols and thus only allows positioning in valid areas of the frame. No verification is done for the body posture being composed. The issues involved in such verification and possible methods for implementing it are explained in Chapter 6. Reselecting a menu option for which a symbol has been previously defined causes replacement of the symbol with the new symbol.

Each body limb is considered as a single unit consisting of a maximum of two symbols defining the extremity and the bend (knee or elbow) positioning. Specifying the bend position is optional. In its absence, a default position is assumed. To position a body limb the editor displays a symbol menu containing all symbols for the extremity and the bend. If the user selects a bend symbol, the system will prompt again for extremity positioning by displaying a symbol menu containing only symbols for the extremity. However, if the user selects an extremity symbol, the editor ignores bend positioning.

The above discussion so far has involved symbols that have been pre-drawn and displayed for user selection. This approach is possible due to the very small number of symbols involved. However, the positioning of the main body, consisting of the head, torso and the pelvis, requires a very large number of symbols (see Chapter 3 on Benesh Movement Notation). It is difficult to display all the possible variations of these symbols at the same time. For this reason, main body symbols are constructed whenever required.

To construct a main body symbol a window containing a default position symbol is displayed as shown in Figure 5.11. The user can now change the symbol by selecting and manipulating the individual line segments. If the longer line is selected, the shorter line disappears and the x-component of all cursor movements drags the top end of the longer line along the top edge of the window while the bottom end of the line is anchored at the mid-point on the bottom edge of the window. The angle between this longer line and an imaginary vertical line passing through the anchor point defines the degree of right or left tilt in the main body part. Pressing the *do* button selects the line position as displayed and a shorter line appears intersecting the longer line in proportion corresponding to the previous positions. If the shorter line is selected, the y-component of all cursor movements is mapped to move the shorter line along the longer line. The x-component of all cursor movements is used to represent a change in the length of the shorter line. This gives a *vernier* effect along the x-axis, allowing users to accurately manipulate the length. The point of intersection of the shorter line with the longer line defines the degree of forward or backward bend while the length of the shorter line defines the amount of turn in the main body part. Again, pressing the *do* button freezes the displayed position. Once the desired symbol has been constructed, selecting the *done* option within the window copies the constructed symbol to its appropriate position in the working frame. Reselecting the main body part option will copy the corresponding symbol from the working frame into the window for further manipulation.

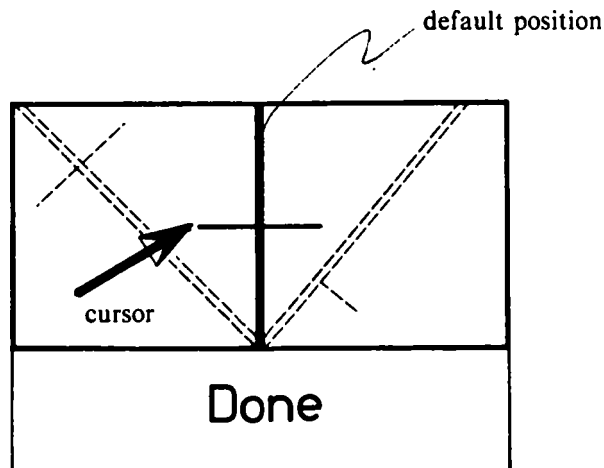


Figure 5.11. Window for constructing main body symbols.



### 5.2.3.2. Spatial Resolution and Visual Discrimination

The size of a menu item is very important for legibility. Bigger menu items are more legible than smaller items. The required size is a function of the screen resolution, the total space available for the menu display, and the number of different options to be displayed in the menu.

The distance between individual options within a menu also affects the selection process. The greater this distance, the longer it takes for the user to move from one option to the neighbouring one, but the smaller this distance the more difficult it will be to visually discriminate between adjacent options during selection. The designer has to find a suitable compromise between these extremes. A simple way to improve visual discrimination between items is the use of bounding boxes which surrounds the menu labels. The menus shown in Figure 5.6 and 5.7 illustrate the use of bounding boxes. Bounding boxes must be used with care so as not to change the semantics of the menu label, especially with iconic labels that may have a meaning beyond the context of the application at hand. An example of this phenomenon from [Dreyfuss 72] is shown in Figure 5.12. The heavy dot means "human" in standard architectural symbols. Adding a bounding box to it resembles a scaled "train" symbol.

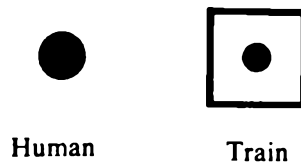


Figure 5.12. Standard architectural symbols [Dreyfuss 72].

Symbol menus are displayed in a circular organization as shown in Figure 5.13. The radius chosen for the circle is a function of the number of symbols in the menu, their size and the spatial distance required between them for visual discrimination. All symbol menus used by the editor have been restricted to contain a maximum of nine symbols.

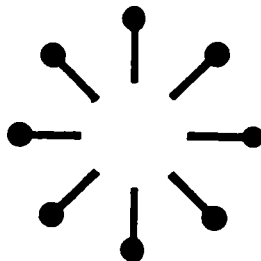


Figure 5.13. Menu of direction symbols.

### 5.2.3.3. Text versus Iconic Labels for Menu Items

*"An idea in the highest sense of that word. cannot be conveyed but by a symbol."*

*- S.T. Coleridge*

We live in a world of graphical images and receive most of our information through visual media, such as television and movies. In some sense, pictures seem to be a "natural" vehicle for communication as they closely resemble the objects to which they refer. By contrast, the written language is based on conventionalized and culture-bound symbolic codes. This argument is generally used to justify the creation of a purely *iconic* language that is not dependent on any written language or culture [Modley 47; Mead 68; Kolers 69; Bliss 65; Huggins 74]. The use of such a non-verbal symbolic language can be a very useful tool for man-machine interaction [Johnson 70; Huggins 69]. However, it is not clear whether a picture can always or even sometimes substitute for a linguistic statement.

Natural language is an excellent medium for communicating propositions intended for people. A picture can also be thought of as conveying a proposition, but the problem with a picture is that it often conveys an infinite number of propositions [Gombrich 72]. Without some constraining context, it is difficult to read the intended proposition. This problem of multiple meanings in a picture has its roots in the very nature of the *human conceptual apparatus*, which uses a finite set of general cognitive frameworks or schemata to form definite descriptions of specific inputs. Even in purely visual perception, the same input can give rise to many different descriptions by drawing on different combinations of schemas depending on changes in context [Mills 81, 82]. This flexibility could create problems for the use of *icons* if they are not designed properly. The design of iconic representations is a specialized area of study that draws on many other disciplines, such as art and psychology. A good icon needs simultaneously to be a *symbol* - something that represents something else by convention or association, and a *picture* - a visual representation. It needs to use abstract graphical elements to highlight generic qualities, but be recognizable as a visual representation of a specific kind of object, situation or event. Poor icon design will inhibit communication rather than help. Dreyfuss presents a collection of standard symbols used internationally [Dreyfuss 72]. These symbols can be easily adopted to icons in man-machine dialogues.

Translating a verbal statement into an equivalent pictorial description is often not easy. In fact, it is difficult to form pictorial propositions about past, future and conditional events or to form logical chains of inferences [Mills 81]. However, in an appropriate combination pictures and text can function together in directing the user towards the intended meaning of a message.

These results have been used in labeling menu items for the editor and in defining the iconic cursors explained in a later section. Abbreviated words from the Benesh terminology have been adopted to label most menu items. Iconic representations have been used whenever available. For example, actual symbolic representations have been used in all symbol menus instead of the conventional symbol names. The body menu in Figure 5.14, showing the body limbs and main body parts, is an excellent example of pictorial labeling of menus. This combination of icons and abbreviated text labels is intended to constrain the label meanings to avoid ambiguity.

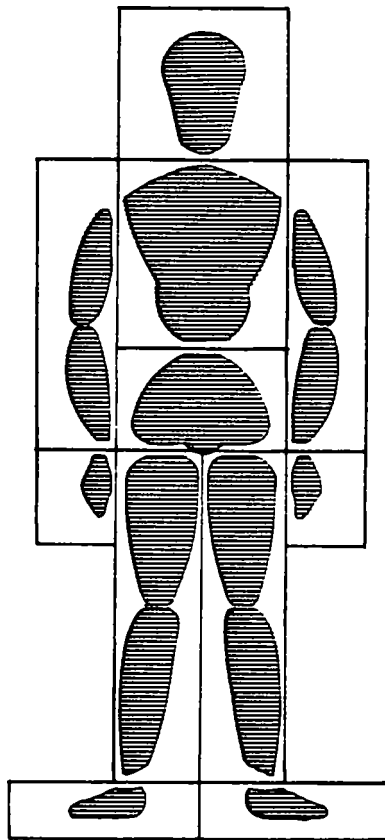


Figure 5.14. Body menu.

#### 5.2.3.4. Dynamic versus Static Menus

Menus that are always displayed at a fixed location when invoked are known as *static menus*. The user becomes accustomed to their display locations and habitually looks for and expects them to appear at the same position. This provides visual continuity and maintains a sense of “place” in the interaction dialogue. However, a drawback of static menus is that the user has to move his eyes (and the hand controlling the puck) from his current position on the display screen to the menu display lo-

cation. In contrast, a *dynamic menu* appears at the user's current position on the display screen, thus avoiding any eye or hand movements to the menu display area. The assumption is that the user's centre of attention is near the *tracker*, his current position on the display screen. Thus a dynamic menu minimizes overall hand and eye movement. Static menus allow the use of tactile memory to pre-position the tablet puck, while the dynamic menus are normally displayed at the user's current position to minimize visual search [Foley 82a].

Dynamic menus are more flexible; they can overlay existing data and so can be displayed anywhere on the display screen. Static menus use fixed display regions that cannot be used for displaying any other data. The total number of static menus that can be displayed is thus limited by display real estate. The static menus provide a convenient way of leaving behind the current system state instead of displaying it at a fixed location.

The editor uses a combination of both types of menus. All symbol menus are dynamic, thus minimizing hand and eye movement for symbol selection. The remaining menus are static, highlighting the *closure* [Foley 82b] of each operation by requiring the user to return to the menu display area. In addition, the static menu structure is also intended to show the system state, as explained in a later section.

#### 5.2.4. Consistency

Consistency asserts that all mechanisms should be used in the same way wherever they occur. This implies that the conceptual model, semantics, command language syntax and display formats should be uniform and not have any exceptions or special cases. For example, the *yellow* puck button is used for selecting the desired symbol from a symbol menu. Thus the same button should always be used for selecting menu options and selecting frames. Consistency is an admirable goal, but a very expensive one to achieve. System complexity for implementation increases with adherence to the consistency principle. Consistency allows the user to generalize the experience and knowledge gained about one aspect of the editor to other aspects. It also helps to avoid the frustration which often occurs when a system does not behave in an understandable or logical way.

The editor's functions are completely consistent within different modes of editing such as manipulating frames, editing within a frame and constructing symbols. However, across these modes not all functions are consistent. Selection is consistent throughout the editor. To select a symbol, frame, or menu item the user aligns the tracker with the desired item and presses the *do* button on the puck. However, the positioning operation is not consistent for positioning frames and symbols. To position a symbol while editing a frame, the tracker assumes the shape of the selected symbol and is then dragged to its final position. To be consistent with this scheme in moving frames, once the frame

sequence to be moved has been selected, the tracking cursor would have to change its shape to that of the selected frame sequence which could be then dragged to its final position. Instead, due to the time constraints for implementing the editor, the user selects the insert position and the editor moves the selected sequence of frames to this position.

One could make positioning of symbols consistent with the positioning scheme used for frames. This scheme was tried and abandoned in favour of the continuous visual feedback provided by dragging.

### 5.2.5. Simplicity

*"Order and simplification are the first steps towards the mastery of a subject - the actual enemy is the unknown"*  
- T. Mann

Simplicity is a very desirable goal. A simple system is better than a complicated one if both provide the same function. One way to make a system appear simple is to make it consistent. This leads to a simple conceptual model which is easier to understand and work with than a complex one. For the most part the editor's functions are consistent. Each function has been broken up into smaller tasks which are independent and very simple. For example, the *move* and *copy* operations have been implemented using the simpler operations *delete*, *save* and *put*. There are a few exceptions to the uniformity and consistency principles because of limitations in resources.

Another way to achieve simplicity is to minimize redundancy in the system. Having two or more ways to do something increases complexity without increasing capability. Too much redundancy may create confusion for users. At the other extreme, a system could have a minimum set of powerful commands that contain all the desired functionality and that do not overlap. Such a set is called *orthogonal*. A designer has to balance between the two extremes. The editor has a minimal redundancy in its operation set. The only operations that contain some redundancy are *move*, *copy*, *delete*, *save* and *put*. As explained in Chapter 4, the *move* and *copy* operations are implemented using the other three operations. This redundancy has been deliberately introduced to simplify the operations and to allow multiple copying of the same selected frame sequence.

Another way to have the system appear simple is to make each of its parts simple. In particular each part should be kept conceptually clean and separate. Sometimes this involves a major redefinition of the user interaction. An example of this situation is the construction of the main body symbol. The interactive techniques used for this construction were complicated when first designed.

The symbol was constructed in three pre-defined stages. First the angle of the long line segment had to be defined. Then its intersection point with the short line was selected, and finally the length of the short line was determined. This sequence required a total of three button pushes to complete construction of a symbol, and the user could not manipulate any single variable without going through the entire sequence. While this scheme was simple to implement it was not simple to use. The newer technique described earlier allows independent manipulation of each symbol variable. It was more difficult to implement but is much easier to use.

### **5.3. Display Representation**

The display is the most visible part of any system and must be carefully designed to be both appealing and informative. It should always reflect the user's conceptual model and should hide all implementation details. The design goal was to minimize the short-term memory required for system operation, so as to relieve it for more important user tasks. This can be achieved by displaying all the information necessary for the user to perform a task at any given time. The editor provides this information through immediate feedback for all user requests, by displaying the system state at all times, and by always displaying the current status of the finished score as well as that of the frame being composed. This section describes the methods used for displaying this information.

#### **5.3.1. Feedback**

Feedback is an essential part of any conversation, whether with a machine or with a person. In a normal conversation with another person, several forms of feedback are exchanged automatically without any conscious action by the participants in the conversation. This feedback includes gestures, body language, facial expressions and eye contact. Foley explains the psychological blocks that occur in the absence of adequate feedback [Foley 74, 82a, 82b]. The most typical symptoms are boredom, panic, frustration, and confusion. Immediate feedback not only avoids these psychological blocks but also provides continuity in the dialogue.

There are three possible levels of feedback corresponding to the levels of the interaction language: lexical, syntactic, and semantic as shown in Figure 5.15 adopted from [Foley 82b]. The editor provides feedback at all of these levels. Lexical feedback consists of changing the cursor icon to the icons shown in Figure 5.16, depending on the puck button pushed. The icons have been adopted from [Dreyfuss 67] and use a combination of text and symbols. An alternate set of icons shown in Figure 5.17 was originally used but later abandoned because of their device dependent association and the rastering effects on diagonal lines.

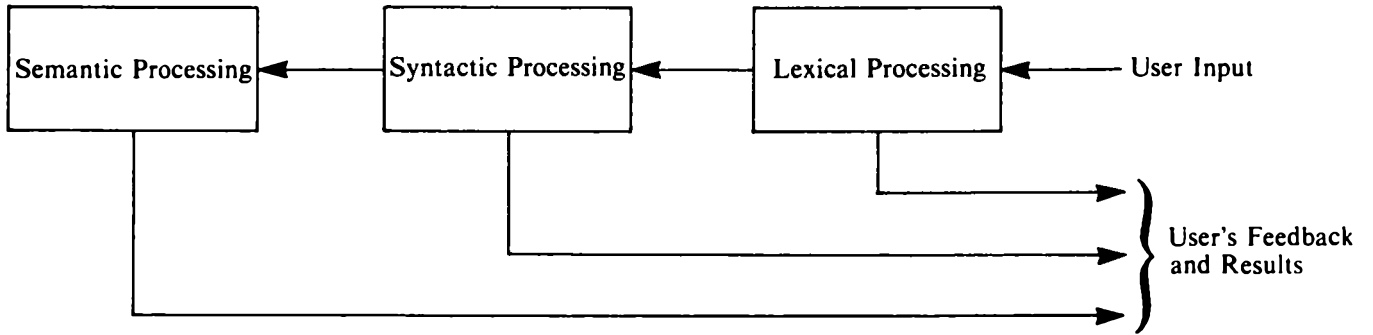


Figure 5.15. Different levels of feedback [Foley 82b].

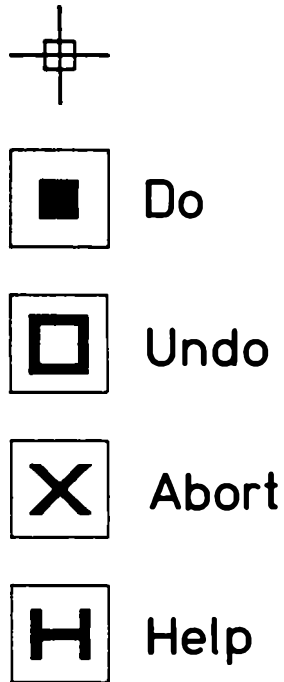


Figure 5.16. Icons for lexical feedback.

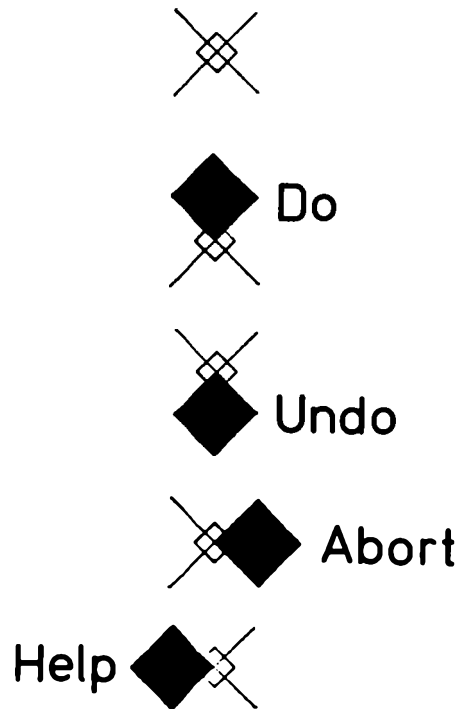


Figure 5.17. Alternate set of icons for lexical feedback.

At the syntactic level, the feedback differs depending on the item being selected. Selecting a frame results in a change in its background colour. Selecting a symbol from a symbol menu replaces the cursor icon with the selected symbol for feedback as shown in Figure 5.18. Finally, semantic feedback is provided by displaying the results of the selected operation.

This feedback hierarchy offers several advantages. Lexical feedback provides a placebo whenever the system is running slowly. However, this placebo can be annoying under normal system operation. When the transition from the lexical to the syntactic level takes a very short time, the lexical feedback turns into an irritating flash. These flashes are especially annoying for the *do* button, the most frequently used button. To avoid this effect, lexical feedback for the *do* button was suppressed and for consistency all lexical feedback has been removed. Thus the current version of the editor provides feedback at the syntactic and semantic levels only.



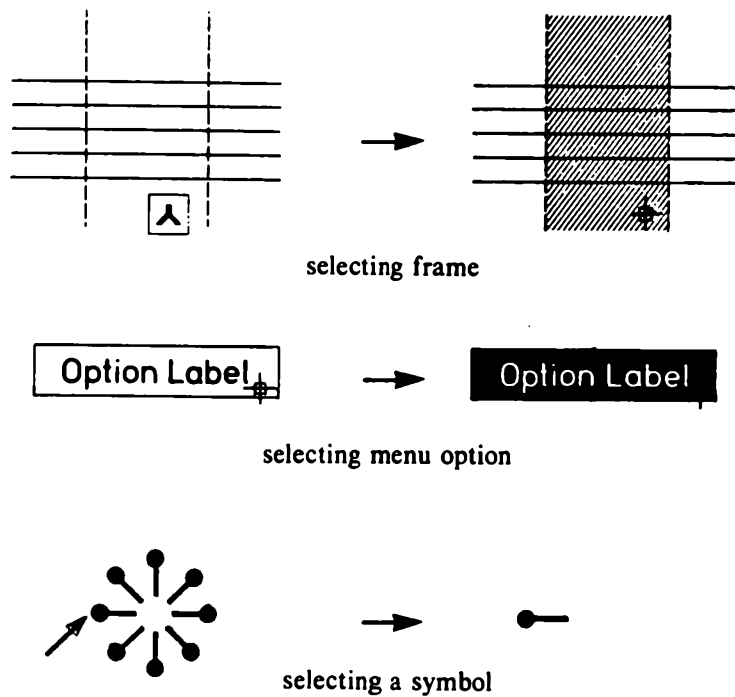


Figure 5.18. Syntactic feedback.

Some form of placebo is still necessary for time-consuming operations such as archiving and dearchiving finished scores. All editor operations have been timed and a placebo is provided for lengthy operations. The placebo used is the *Buddha* icon, adopted from Newswhole [Tilbrook 80], as shown in Figure 5.19.



Figure 5.19. Buddha icon [Dreyfuss 67].

### 5.3.2. The System State

A very important aspect of the display representation is showing the system state at all times. Most systems reserve an area of the display screen for displaying this information. However, there are some drawbacks to this approach. Viewing television is very different from viewing the editor's display. In the former case, it is a one way communication and the viewer sits back and relaxes. His *centre of attention* is the whole screen and if a message suddenly appears at the bottom of the television screen, he notices it. But in the case of the editor's display there is a two-way communication: the user manipulates the objects that he sees on the screen and the system acknowledges user prompts with appropriate feedback. During an interactive dialogue, the user's centre of attention is generally focussed on the object that he is manipulating. Anything appearing outside his centre of attention, especially outside his peripheral field of vision, will not register. Therefore the problem with displaying the system state in a fixed location is that if this location falls outside the user's centre of attention but within his peripheral field of vision, it will distract his attention. The user's curiosity will move his centre of attention to the location where the system state is displayed. After reading the state information, the user will return his attention back to the original position. This eye movement every time the system state is changed can be very tiring and very annoying, especially if the system state changes quite frequently. A better way to present this information would be to display it within the user's centre of attention, thus avoiding eye movement and providing visual continuity.

The system state referred to here is not the internal program state, but the current mode of the editor within the user's conceptual model. Selecting an option in a menu other than a symbol menu inverts the option label explained earlier in the last section and all other options are switched off. This inverted menu option label shows the current mode of editing. As the user moves along the menu hierarchy by selecting menu option, he leaves behind a trail of inverted option labels, the most recent of which is always at the centre of attention. Together, these labels indicate the specific state of the editing process.

The editor also shows the action in progress by changing the tracker icon, as shown in Figure 5.20. The editing mode defined by the inverted labels and the current action in progress as shown by the iconic tracker together define the complete state of the system at a given time.

### 5.3.3. The Current Status of the Score

The editor displays the finished score in the display window. While editing a frame, the selected frame is left highlighted to indicate its context within the score. When adding new frames to the finished score, the editor displays the current insertion position, which is updated each time a new frame is added. All frames added during the current *add frame* operation are left highlighted until

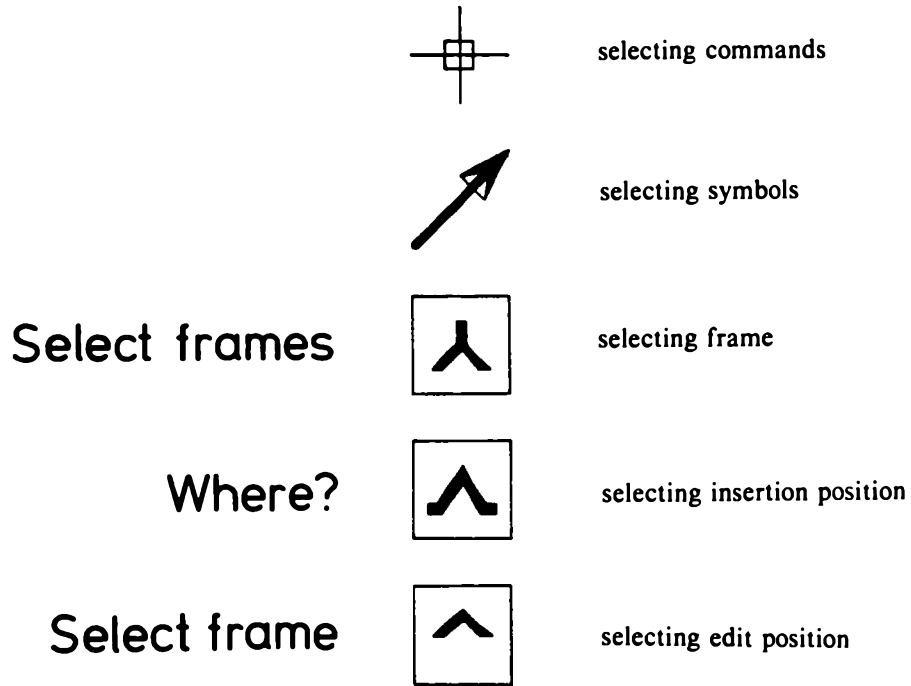


Figure 5.20. Tracker icons.

the end of the operation. This always shows the current status of the finished score. However, only the frames within the display window are shown at any given time and the user has no indication of the relative position of the display window within the score. A *scroll bar*, explained in Chapter 6, could be used to indicate the display window position and to manipulate this window.

When editing a frame or composing a new frame, the working frame shows three different types of information. Firstly, a body part may not have been defined yet. Secondly, a body part may have been carried over from the previous frame. Thirdly, a body part may have been newly defined for this frame. All body information is shown using symbols. One way to show the different types of information is to use symbols of different colour. However, this would imply an ordering of symbols by their importance and thus change the semantics of the symbols. This would also require more than one bit-plane in the foreground. Instead the editor displays this information for each body part by using different background colours for the corresponding menu items, as shown in Figure 5.21. A black background means that the menu item has not been selected. Colour association is used for the remaining types of information. If the background colour of a menu item is the same as the highlight colour of the last frame added to the finished score, this implies that the corresponding symbol has been carried over from the previous frame and selection of this item will result in replacement of the existing symbol. A background colour which is the same as that of the working frame implies that

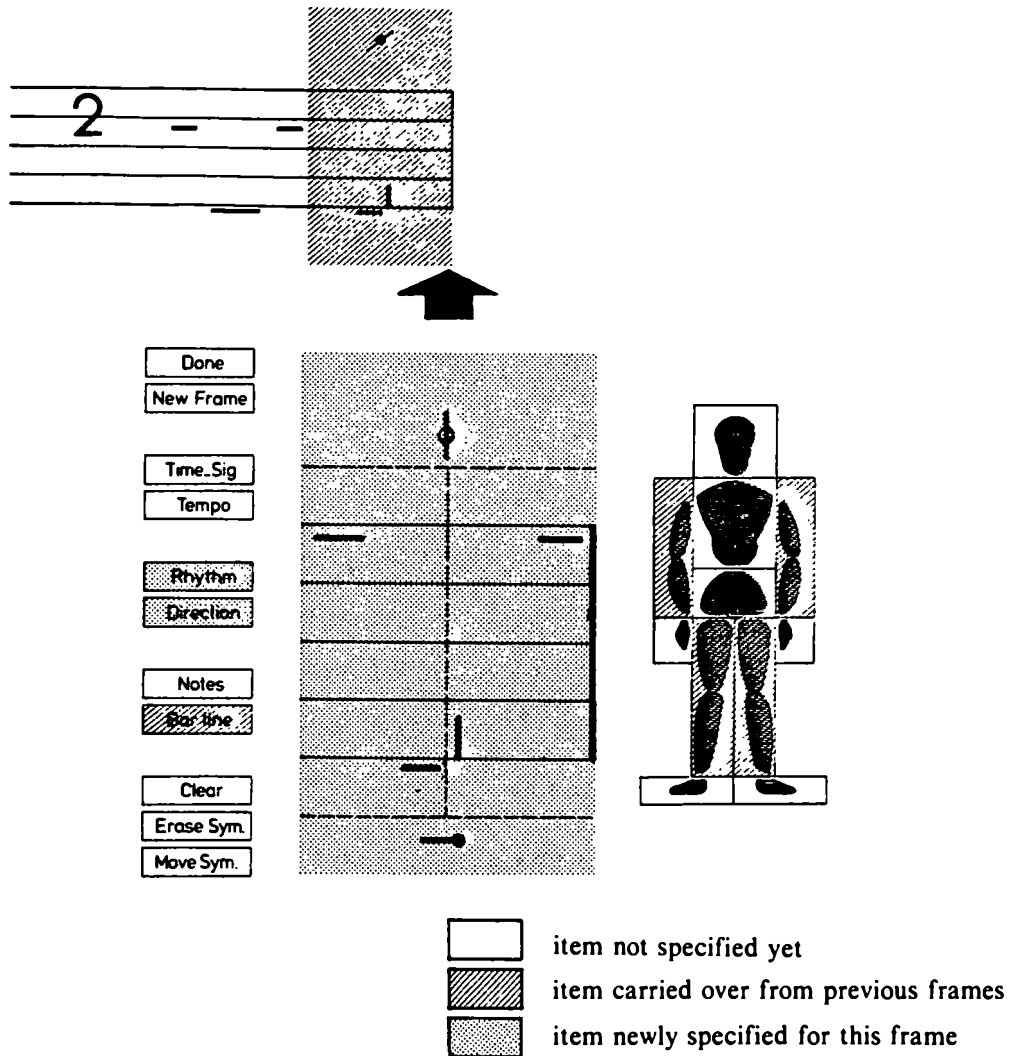


Figure 5.21. Status information display.

the corresponding body part has been newly defined. This colouring scheme provides a checklist of body information that has been defined and that needs to be defined.

This chapter has touched on some key areas of the user interface design through a description of the interface design and implementation for the Benesh editor. There are other issues which have been considered but not implemented due to lack of time. These ideas are the topic of the next chapter.

## 6. FUTURE EXTENSIONS

*"It is a bad plan that admits no modification."*

*- P. Syrus*

When computerizing an existing system a number of new capabilities are possible which would otherwise be impossible or not worthwhile to implement. The functions provided by the Benesh editor are very basic and limited by the conceptual model, which is identical to the basic Benesh model. However, extending this model to take advantage of the possibilities inherent in a computerized system results in some very useful capabilities that can enhance the process of editing Benesh scores. This chapter describes extensions to the editing functions, movement verification, hard copy facilities, and potential improvements to the editor's user interface.

### 6.1. Extension to Editing Functions

Chapter 4 described the analogy between text and Benesh. This analogy suggests the full range of text editing facilities for editing Benesh scores. The following is a list of desirable features to further increase these editing capabilities.

#### Multiple Buffers and Windows

When composing a score manually, one easily switches back and forth between different tasks such as writing one score while correcting another, or comparing several scores at the same time. This ability can be provided in the Benesh editor using multiple buffers and multiple windows. As in Fred [Gardner 80] multiple buffers store each score separately and allow the user to switch back and forth between them at will. Any changes that have been made to any of the scores remain, even if the user switches from the buffer and then back again. Multiple windows split the display screen so that two or more scores can be displayed at the same time. The System Product Editor allows user to divide the physical screen window into multiple windows to edit multiple files or view different segments of the same file [IBM 70]. Each window is in effect, an independent terminal with its own file identification line, command line, status area, and message line. Using this technique, different parts of the same score could also be displayed simultaneously.

### Higher Level Score Objects

The full-screen editor *Vi* on Berkley Unix deals with high level text objects such as sentences, paragraphs, and sections [Joy 80]. Benesh notation also consists of high level objects such as a *bar*, corresponding to a bar of music, or a *movement section*, consisting of all frames sharing the same time signature. It is often advantageous to work in terms of these objects. Some useful operations at this level would be *move*, *copy*, *delete*, and *print* (hard copy output). The implementation of these operations requires a *search* for the beginning and end of a bar as well as for the previous and the next header frame. Using this search function operations for high level objects can be mapped into basic frame operations.

### Full-screen Display Window

The sixteen frame display window is very limited. A full screen scrolling window would be very helpful, especially when reading the finished score. Existing capabilities for manipulating the display window are limited and would need to be expanded to include such facilities. User feedback about the location of the display window within the score is currently inadequate, as explained in Chapter 5. A *scroll bar*, as shown in Figure 6.1, gives a simple way of manipulating the display window and also provides a means of feedback showing its position within the score. A scale ranging from zero to the maximum number of frames in the score, displayed below the scroll bar, could be used for additional feedback. The dark, rectangular region of the scroll bar is the *scroll marker*. Its relative position within the scroll bar corresponds to the position of the display window within the score. The length of the marker corresponds to the actual size of the display window. The user should be able to control this size through the number of frames to be displayed in the window. To move the display window along the score, the user would select the marker and move it along the scroll bar. As the marker moved across the scroll bar, the display window would show the corresponding section of the score.

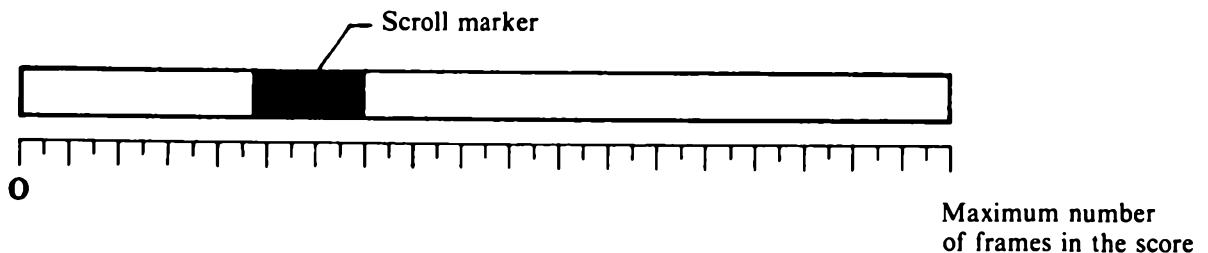


Figure 6.1. Scroll bar.

### Marking and Returning

A frame position can be identified by assigning a unique name to it known as a *mark*. If a list of all marks is displayed in a fixed location, the user can return to a marked position by simply selecting the corresponding mark. This provides an efficient means for moving to a specific frame within the score and for specifying operands for frame operations. These marks can also be used as operands to commands, instead of pointing to the actual frames. Furthermore by considering each frame as marked by a number representing its position within the score, the user can specify operands by pointing along the scroll bar scales. If a marked frame is deleted, the corresponding name is erased from the display. The user should also be able to unmark a frame position. Marks are specific to a score and can be archived and de-archived along with the score.

### Macros

The idea of marking a frame can be extended to identifying a sequence of frames, one form of the *macros* commonly used in computer science. Invoking a macro would insert the corresponding sequence of frames after the current frame position. This avoids the need to recompose or copy commonly used sequences of frames each time they are required. One could build a whole library of such useful macros and build a score entirely from these macros. For example, most of the common movement sequences in dance have been assigned specific names which are part of its vocabulary. These movement sequences can be declared as macros and retrieved from the library without the trouble of recomposing them. These macros can be made more flexible by using user supplied parameters to change the contents of the macro, similar to commonly used *macros with parameters* in computer science.

### High-level Operations

The features discussed so far are not specific to Benesh notation but can be used for editing any kind of information. This section describes some additional features that are specific to movement notation.

Often one would like to perform operations on movements, such as *changing* the dynamic qualities of movements and *transforming* movements. Changing the dynamic qualities of movements refers to operations in the time domain, such as accelerating or decelerating movements. This amounts to redefining the *tempo* and *degree of effort* information in a Benesh score. A typical movement transformation is *reflection* - changing all body part movements into their mirror movements

with respect to the sagittal plane (see Figure 3.1). The user feedback for these operations requires new concepts and techniques, explained in the next section on movement verification. The editor currently handles notation only for a single dancer. Multiple dancers introduce additional transformations such as assigning the movements of one dancer to another or changing the direction of one dancer with respect to another. The operations described here can be applied either to the whole body or to an individual part of the body.

Another desirable feature is to identify movement sequences in a ballet as essential or optional. This can be done by colour coding the Benesh frames. In addition, alternate movement sequences can also be specified using this method.

### Options

All users may not require the full functional capabilities of the editor. Therefore, it is advantageous to provide only the basic editing functions and to organize the remaining capabilities as options invoked on request. The editor's mode of operation is now defined by the combination of requested options. A useful editor option would be *read only*. In this mode the editor would display the score for reading but not allow any editing. Another useful option would be the verification of notated movements, as described in the next section. Different editor modes could be used to create an editing environment suitable for the user's level of experience. The editor could guide new users by displaying all relevant help information, while for experienced users this information could be suppressed. This allows the editor to cater to the needs of individual users and to make efficient use of the available resources.

The features currently available and the ones presented here require an efficient memory management scheme for their implementation. One approach is to organize the complete score as a set of display pages and keep only the current page plus the two adjacent ones in main memory while storing the other pages on secondary storage. When a required page is not in core, the pages in core are moved to the secondary storage and replaced by the new pages. The techniques for implementing such facilities are well known and available in most text editors.

## **6.2. Movement Verification**

One of the most attractive features of computerizing the editing of Benesh scores is the possibility of verifying the notated score. One approach is to verify each body posture within a movement sequence as it is being composed and to prompt the user to correct errors, if any are detected. This requires a body model against which all notated information can be checked for possible errors. Such a body model is described below.



## Body Model

The *skeleton* is the basic structure to which all information from the notation is related. It consists of straight line segments in three-dimensional space, each corresponding to a body part as shown in Figure 6.2. The intersection points of these straight line segments correspond to the body joints. The main advantage of this structure for representing human movements is its simplicity. It can be expanded to increase its flexibility by simply breaking the straight line segments to create new joints and by adding more segments. Thus the upper torso could be modeled more accurately by a sequence of short line segments corresponding to the spine.

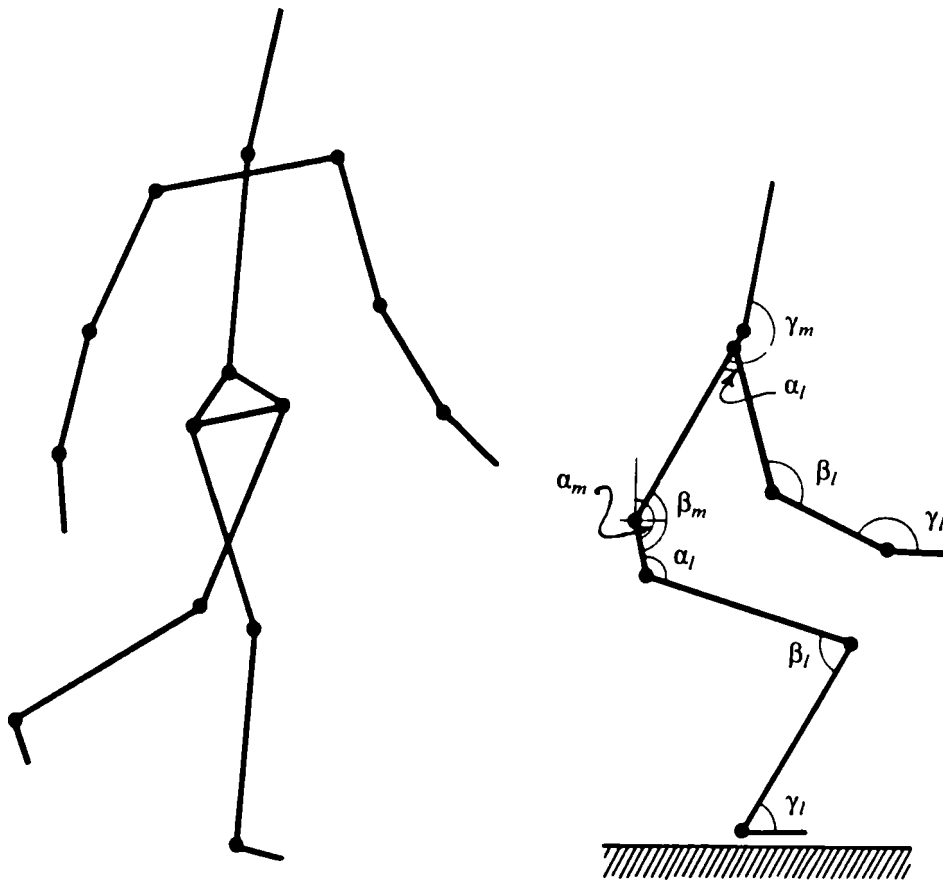


Figure 6.2. The skeleton.

Each angle has three components, one in each of the coronal, sagittal and transverse planes

$\alpha_m$  - the angle of the pelvis to the body mid-line.

$\beta_m$  - the angle of the upper torso to the pelvis.

$\gamma_m$  - the angle of the head to the upper torso.

$\alpha_l$  - the angle of the upper arm to the shoulder or the angle of the upper leg to the pelvis.

$\beta_l$  - the angle between the upper and lower limb parts.

$\gamma_l$  - the angle of the hand or foot to the lower limb parts.

The skeleton is divided into two major parts, the *main body* and the *limbs*. The main body is divided into *head*, *upper torso*, and *pelvis*, while the *left arm*, *right arm*, *left leg*, and *right leg* constitute the limbs. Each arm is further sub-divided into the *upper arm*, *lower arm*, and *hand*. Similarly each leg is sub-divided into the *upper leg*, *lower leg*, and *foot*. Detailed information about a body position can be compiled by noting the relative angles between these straight line segments in the *sagittal*, *coronal*, and *transverse* planes (see Figure 3.1). These angles are measured between adjacent straight line segments in order of inferior (lower) to superior (upper) body parts for the main body and superior to inferior body parts for limbs (see Figure 3.1). For example, the main body position is recorded as a tuple  $(\alpha_m, \beta_m, \gamma_m)$ , where  $\alpha_m$  is the angle of the pelvis to the body mid-line, which is perpendicular to the floor,  $\beta_m$  is the angle of the shoulder to the pelvis, and  $\gamma_m$  is the angle of the head to the torso. Similarly each limb position is recorded as a tuple  $(\alpha_l, \beta_l, \gamma_l)$ , where  $\alpha_l$  is the angle of the upper arm to the upper torso or the angle of the upper leg to the pelvis,  $\beta_l$  is the angle between the upper and lower arm or leg, and  $\gamma_l$  is the angle between the palm or foot and the lower arm or leg respectively (see Figure 6.2).

The body limb positions are recorded in relation to the main body parts, irrespective of the position of these parts. The position of the main body parts is in turn recorded in relation to the body mid-line. The positions of limbs are depicted by their projections onto the frontal plane. This projection onto the stave provides only two-dimensional information. To extract the third dimension it is necessary to know whether the projected point is in front, behind or within the frontal plane. This information is given by the basic sign used to plot the projection point.

In mathematical terms, the problem is to calculate the defining angles for a three-dimensional vector in the  $x$ - $y$ ,  $y$ - $z$ , and  $x$ - $z$  planes. The length of the vector and its projection onto the  $x$ - $y$  plane is known. The solution is shown below:

Let the vector be  $(x, y, z)$  with length  $l$ . We know that

$$x^2 + y^2 + z^2 = l^2$$

and knowing  $x$ ,  $y$  and  $l$  we have

$$z = \pm \sqrt{l^2 - x^2 - y^2}$$

Now the angles in the three planes can be calculated as

$$\text{angle in the } x\text{-}y \text{ plane} = \sin^{-1} \left( \frac{y}{\sqrt{x^2 + z^2}} \right)$$

$$\text{angle in the } y\text{-}z \text{ plane} = \sin^{-1} \left( \frac{z}{\sqrt{y^2 + z^2}} \right)$$

$$\text{angle in the } x\text{-}z \text{ plane} = \sin^{-1} \left( \frac{x}{\sqrt{x^2 + z^2}} \right)$$

*Note* :- Knowing the length of a vector and its angle in any two of three orthogonal planes uniquely defines its relative position in a three-dimensional space. These are really just the familiar direction cosines of trigonometry.

The limb positions can now be obtained by applying the above mathematical solution and considering each skeleton limb to be a three-dimensional vector. The lengths of all skeleton line segments are shown in Figure 6.3 and the  $x$  and  $y$  coordinates can be obtained from the location of the basic sign in relation to the pivot point of the limb. In addition, the basic sign used for the body part extremity indicates whether the calculated  $z$  coordinate is negative, positive or zero. When the basic sign is a vertical stroke (*in front*), the extremity is in front of the frontal planes, and this implies that the calculated  $z$ -coordinate must be greater than that of the frontal plane. Similarly, when the basic sign used is a dot (*behind*), the calculated  $z$ -coordinate must be less than that of the frontal plane. Finally if the basic sign is a horizontal stroke, the calculated  $z$ -coordinate must be the same as that of the frontal plane. Benesh symbols provide only an approximation to the projection of each body part. If the projection points were really precise, a  $z$ -coordinate equal to that of the frontal plane would be apparent by the length of the projection being equal to the length of the limb.

The solution given above is very simple, but as pointed by Mendo, there are cases where knowing the projection location on the stave and the length still leaves two possible positions [Mendo 75]. The ambiguity is in the notation, rather than the mathematical solution. Figure 6.4 illustrates this ambiguity using an upper body limb. To remove this ambiguity the convention followed is that the basic signs mark either the only possible body part position or the one furthest away from the body. A new sign has been introduced to cover these situations. It is an open dot as shown below:



position closest to the frontal plane which is between the frontal plane and the furthest possible position.

The ambiguity mentioned above arises only in the presence of bent body limbs. Therefore, the new sign is always used in conjunction with the bend signs, and the basic sign used for the bend determines whether the projection point indicated by the new sign is in front of or behind the frontal plane.

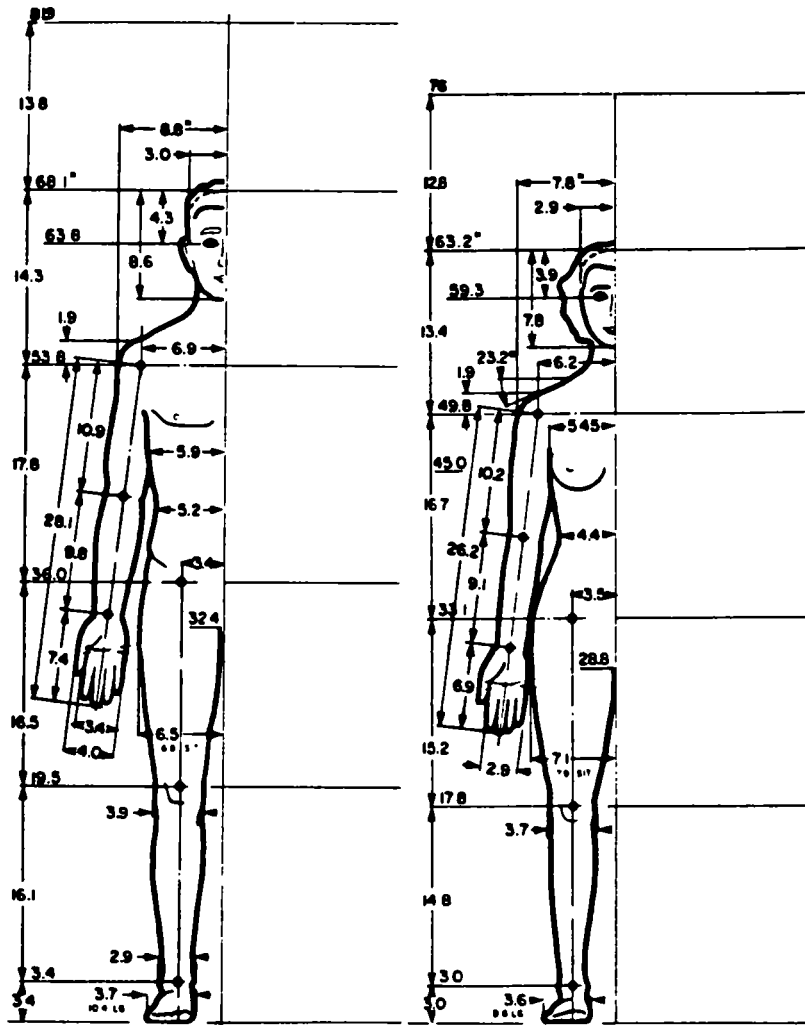


Figure 6.3. Measure of average men and women [Dreyfuss 59].

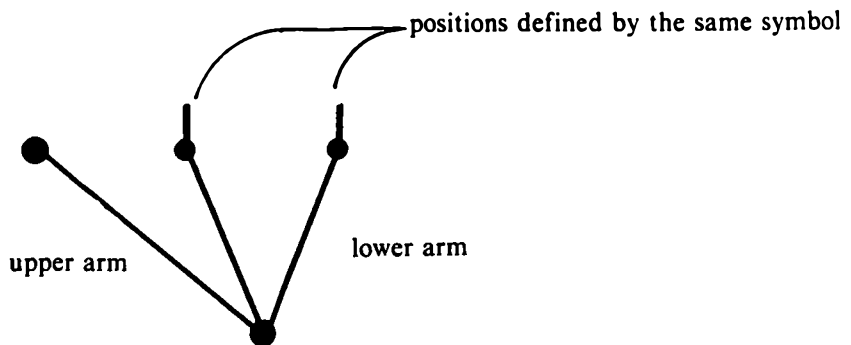


Figure 6.4. Ambiguity in extremity positioning.

The main body part angles are directly defined by the symbols used as illustrated in Figure 6.5. In fact, no verification is necessary for their position because the symbol construction process is constrained to provide only possible positions.

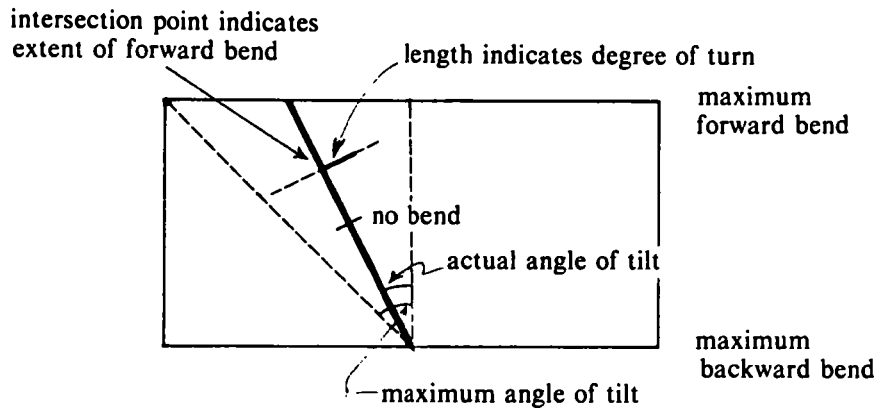


Figure 6.5. Main body symbol.

Knowing the angles  $\alpha_m$ ,  $\beta_m$ ,  $\gamma_m$  for the main body and  $\alpha_l$ ,  $\beta_l$ ,  $\gamma_l$  for each body limb, we can construct the complete static body posture in three-dimensional space. To interpret and verify Benesh Movement Notation, we need to extract this angular information from the notation.

In the Benesh stave, all five lines and the two leger lines are equidistant and intersect the body at specific points as shown in Figure 3.2. However, human engineering data charts [Dreyfuss 59] show that the distances between the five intersection points of the stave lines with the human body are not equal. Figure 6.3, adopted from [Dreyfuss 59], gives body measurements for an average man and woman. It shows the exact intersections of the stave lines with the body. Therefore the first step in extracting three-dimensional information from a Benesh score is to convert all the data to a normalized stave with proportions based on the measures shown in Figure 6.3. This enables us to verify the positions of the symbols with respect to the stave lines. The angular information can then be extracted from the normalized stave and compared against anatomical data such as that in [Heck 65] to verify the scored position.

The verification scheme described above can only verify individual body part positions. The movement of a particular body part also has a constraining effect on the possible movements of all other remaining parts in order to maintain overall balance. Therefore, to truly verify a posture, the body model must take into account the centre of gravity. One such model is described in [Hanavan 66].

Another problem in verification that cannot be detected by the model described above or by calculating the centre of gravity is the collision between body parts. With a stick-figure model, this collision detection amounts to determining intersections of the underlying line segments in three-dimensions of space. However, collision detection with solid models is more complicated. One approach is to divide the body parts into volumetric objects. Now the collision problem can be reduced to finding the volumetric intersection between all these objects. If the result of such an intersection is null, then there are no collisions. If there is a collision then we know the body parts involved in this collision. Since the geometric body representation can be very complex, finding all intersections could be very time consuming. One possible strategy is to use two representations of the body parts. For example, a crude approximation by bounding boxes (see Figure 6.6) and an accurate representation with a larger number of more complex solids. Initially we calculate the intersection using the crude representation. If a collision exists using this representation, then we perform the more accurate second level of intersection calculation only for those parts which were involved in a collision.

Another aspect of this problem is collision between dancers. In this case a single bounding box for each dancer could be used as a very rough approximation to determine any pairs of dancers which might potentially collide. Again more detailed calculations would only have to be performed on critical pairs.

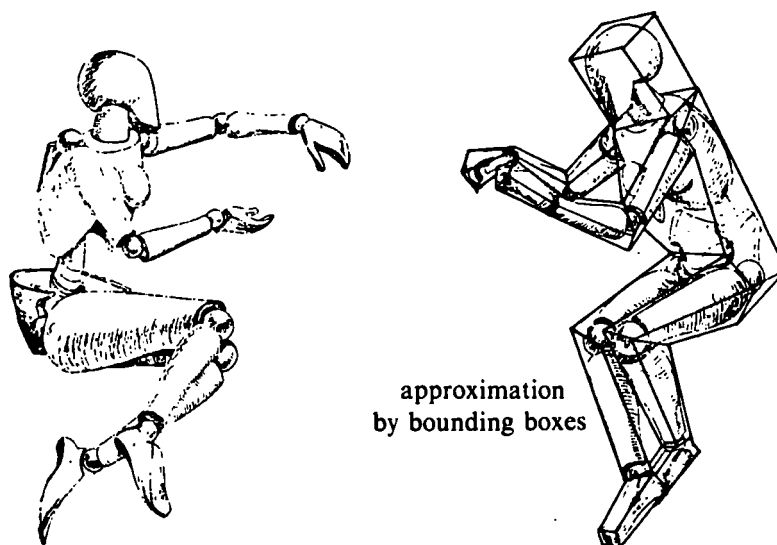


Figure 6.6. Human body representations using solids.

Any complete verification requires a final visual check of the actual notated movement. Each frame in Benesh Movement Notation defines what is known to film animators as a *key-frame*. An animation system with a suitable interpolation algorithm could be used to calculate the intermediate positions between Benesh frames. This would enable us to verify movements by producing a stick figure animation from the sequence of Benesh frames and the intermediate frames [Calvert 82a]. Such interpolation techniques for generating smooth animated motion are discussed in [Kochanek 82].

The body models discussed for movement verification can also be used for translating movement scores between various notation systems. For example, the complete three dimensional body information calculated from the Benesh score using the skeleton model can be used as a basis for this translation. Similar information is collected by other systems based on Labanotation and the reverse mapping from body information into this notation also exists [Calvert 82a].

### **6.3. Hard Copy Facilities**

Once the score has been created, modified and verified it is ready for printing. The printed copy can then be preserved in a library of scores for later recreation. The reason for maintaining a library of hard copy scores is purely historical. More compact methods of storage are available such as micro-film. In fact, a score can be stored in its digital form and printed only when necessary. The printed score can be used for reference during rehearsals or can be distributed to dancers for learning their individual parts, similar to a script for actors. The final goal of any notation editor is to produce typeset copies of the score suitable for reproduction. Since a Benesh score consists of a limited set of symbols, it is possible to define a specialized Benesh type font. Several manufacturers of phototypesetting equipment allow the incorporation of such user-defined fonts into their system.

### **6.4. Improvements to the User Interface**

A major improvement in the user interface would be the display of a human figure which actually assumed the notated positions. The figure does not have to be very realistic; a stick figure would be sufficient for most purposes [Calvert 82]. This figure could appear in a window on the screen adjacent to the working frame. As the body position is being notated the corresponding body parts of the figure would move accordingly. Conversely, the user could move any parts of the stick figure causing the appropriate Benesh symbol to be positioned in the working frame. Such a scheme might require three-dimensional cues such as perspective and hidden line removal. *Haloing* techniques might be sufficient for showing hidden lines [Beatty 81; Appel 79]. However, the user must be able to change the viewpoint and rotate the stick figure.

Another area for further improvement is the static menu display. Currently the menu displays a limited number of options. As new functions are implemented to improve the system's editing capabilities, new options must be added to the menu. The upper limit for the total number of items that should be displayed simultaneously in a menu depends on the display techniques used. One approach to expand this limit is to use cyclical menus with a view window which displays a section of the menu as shown in Figure 6.7. The items within the menu could be logically grouped as explained in Chapter 5. The number of items within a group should be restricted to about seven [Miller 56]. Menu items within the view window would be visible at a given time. However, the remaining items could be displayed by scrolling the window. A slider provided for this purpose is shown in Figure 6.7. The scrolling speed could be adjusted using the slider provided on the side of window and the user could freeze the window display when the required item is displayed. It is important not to simply freeze the window display immediately, but to check the current position and display the group containing the maximum number of items displayed when scrolling was stopped. If it takes too long for an item to cycle back to the view window, this could cause user frustration. For this reason the total number of items in the circular list should be limited to perhaps three times the number of items that can be displayed in the window.

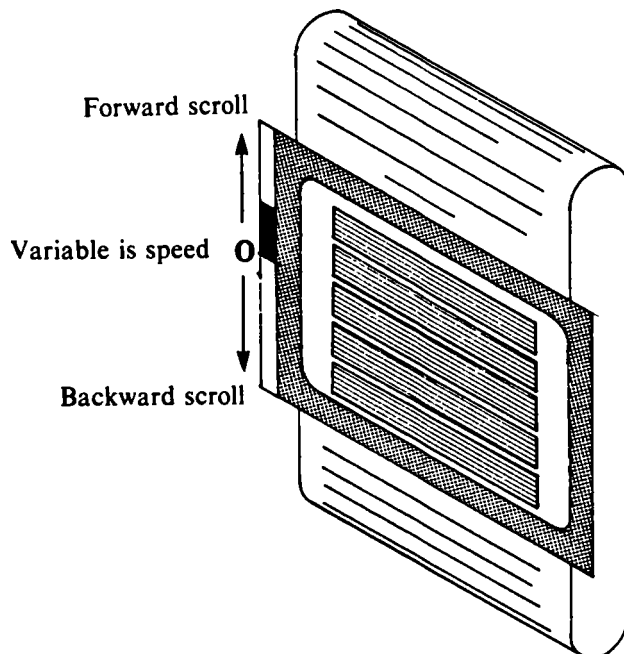


Figure 6.7. Scrolling menu.



The editor uses a four button puck as the primary input device as shown in Figure 5.5. Each button has been colour coded for easy differentiation. However, the user normally doesn't look at these buttons, but uses tactile memory to locate them. It has been observed throughout the prototyping and testing phases of the system that users consistently have trouble selecting the appropriate button for all actions other than the *do* function (yellow button). The colour coding is obviously of no help. A suitable lexical feedback would be helpful but it has other drawbacks as described in Chapter 5. These observations suggest the use of a single button for all functions. This can be achieved by turning the puck button display described in Section 5.2.3 into a pickable menu. Now either a stylus or a puck can be used, making the input function less device dependent.

Implementing the ideas discussed in this chapter would be a considerable amount of work, but the resulting system would very likely save a considerable amount of the choreologist's time.

## 7. CONCLUSIONS

*"It could be one of the most exciting intellectual developments of our time; for the language of creative thought in every domain is notation and the invention of a new notation is indeed the discovery of a new creative tool "*

*- J. Benesh*

The editor described in this thesis is designed to facilitate the learning, teaching, and use of the Benesh Movement Notation. The particular value of such a system lies in its ability to edit any frame or set of frames as often as necessary and to obtain a fair copy of the score after each editing cycle.

Computer scientists have long recognized the need for high level languages to model the abstraction of various concepts without having to be concerned with low level details. The last decade has witnessed the emergence of many high level programming languages. Benesh Movement Notation is a high level language that models our abstraction of human movement. It captures the mode, quality, and rhythm of movements at a fairly high level without dealing with the detailed biomechanical processes involved. However, there are other languages such as Labanotation [Hutchinson 77] and Eshkol Wachman notation [Eshkol 75] that provide more detailed information on movement. Many of the techniques developed in the Benesh editor would be equally applicable to these other notation systems.

Ronald Baecker and William Reeves have developed *p-curves* and *moving points* to define smooth movements in computer animation [Baecker 69; Reeves 81]. The concepts behind these developments are similar to the idea of movement lines and effort in Benesh notation. With some modifications, Benesh Movement Notation could be used to define the movement of animated figures. In addition, it could also be used to specify and verify robotic movements. Thus, while Benesh Movement Notation was designed purely to record ballet movements, it could also provide a high level tool for defining other types of movements such as in animation and robotics. To be useful in these applications there must exist editing tools at least as powerful as those universally in use for text processing. The Benesh editor is a first step toward this reality.

## References

- [Anderson 79] Anderson, J., "Preserving Dances in Print," *New York Times*, Sunday, May 6, 1979.
- [Appel 79] Appel, A., F.J. Rohlf, and A.J. Stein, "The Haloed Effect for Hidden Line Elimination," *Computer Graphics*, 13(2), pp. 151-157, August 1979.
- [Archer 75] Archer, L. B., *A Study of Computer Choreography*, Royal College of Arts, London, 1975.
- [Arnheim 69] Arnheim, R., *Visual Thinking*, University of California Press, Berkeley, 1969.
- [Badler 78a] Badler, N.I., J. O'Rourke, and H. Toltzis, "A Human Body Modelling System for Motion Studies," Movement Project Report No. 13, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, July 1978.
- [Badler 78b] Badler, N.I., J. O'Rourke, S. W. Smoliar, and L. Weber, "The Simulation of Human Movement by Computer," Movement Project Report No. 14, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, July 1978.
- [Badler 79a] Badler, N.I., J. O'Rourke, and H. Toltzis, "A Spherical Representation of a Human Body for Visualizing Movement," *Proceedings IEEE* 67(10), pp. 1397-1403, October 1979.
- [Badler 79b] Badler, N.I. and S.W. Smoliar, "Digital Representation of Human Movement," *Computing Surveys* 11(1), pp. 19-38, March 1979.
- [Badler 80] Badler, N.I., J. O'Rourke, and B. Kaufman, "Special Problems in Human Movement Simulation," *Computer Graphics (Proceedings Siggraph '80)* 14(3), pp. 189-197, July 1980.
- [Baecker 69] Baecker, R.M., "Picture-Driven Animation," *Proceedings AFIPS Spring Joint Computer Conference* 34, AFIPS Press, Montvale, N.J., pp. 273-288, 1969.
- [Barenholtz 77] Barenholtz, J., Z. Wolofsky, I. Ganapathy, T.W. Calvert, and P. O'Hara, "Computer Interpretation of Dance Notation," pp. 235-240 in *Computing in the Humanities, Proceedings of the Third International Conference on Computing in the Humanities*, eds. Serge Lusignan and John S. North, The University of Waterloo Press, Waterloo, Canada, August 1977.
- [Beatty 81] Beatty, J.C., K.S. Booth, and L.H. Matthies, "Revisting Watkins Algorithm," *Proceedings 7th Canadian Man-Machine Communications Conference*, Waterloo, Ontario, pp. 359-370, June 1981.
- [Benesh 56] Benesh, R. and J. Benesh, *An Introduction to Benesh Dance Notation*, A. and C. Black, London, 1956.
- [Benesh 74] Benesh, R. and J. McGuiness, "Benesh Movement Notation and Medicine," *Physiotherapy* 60(6), pp. 176-178, 1974.

- [Benesh 77] Benesh, R. and J. Benesh, *Reading Dance, The Birth of Choreology*, Souvenir Press (E&A) Ltd., 1977.
- [Birdwhistell 70] Birdwhistell, R.L., *Kinesics and Context: Essays on Body Motion Communications*, University of Pennsylvania Press, Philadelphia, 1970.
- [Blau 80] Blau, R. and J. Joyce, "Ex Reference Manual," *UNIX Programmer's Manual, Seventh Edition (VAX II Version) 2c*, Computing Services, University of California, Berkeley, August 1980.
- [Bliss 65] Bliss, C.K., *Semantography: Bliss Symbolics*, Semantography Publications, 1965.
- [Booth 82] Booth, K.S. and S.A. MacKay, "Techniques for Frame Buffer Animation," *Proceedings Graphics Interface '82*, NCGA, Toronto, pp. 213-220, 1982.
- [Bourne 79] Bourne, L.E. and B.R. Ekstrand, *Psychology: Its Principles and Meanings*, Holt, Rinehart and Winston, 1979. (3rd edition)
- [Boysen 77] Boysen, J.P., P.R. Francis, and R.A. Thomas, "Interactive Computer Graphics in the Study of Human Body Planar Motion under Free Fall Conditions," *J. Biomechanics* 10(11/12), Pergamon Press, Great Britain, pp. 783-787, 1977.
- [Breslin 82] Breslin, P., "A Powerful Interface to a High Performance Raster Graphics System," M.Math. Thesis, Department of Computer Science, University of Waterloo, 1982.
- [Brown 82] Brown, J.W., "Controlling the Complexity of Menu Networks," *Communications of the ACM* 25(7), pp. 412-428, July 1982.
- [Brown 76a] Brown, M.D. and S.W. Smoliar, "A Graphics Editor for Labanotation," *Computer Graphics* 10(2), pp. 60-65, Summer 1976.
- [Brown 76b] Brown, M.D., "A Graphic Editor for Labanotation," M.Sc. Thesis, University of Pennsylvania, 1976.
- [Brown 78] Brown, M.D. and S.W. Smoliar, "Preparing Dance Notation Scores with a Computer," *Computers and Graphics* 3(1), pp. 1-7, 1978.
- [Burtnyk 76] Burtnyk, N. and M. Wein, "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation," *Communications of the ACM* 19(10), pp. 564-569, October 1976.
- [Buxton 78] Buxton, W. "Design Issues in the Foundation of a Computer-Based Tool for Music Composition," Technical Report CSRG-97, Computer Systems Research Group, University of Toronto, 1978.
- [Calvert 68] Calvert, T.W., "Projections of Multidimensional Data for Use in Man-Computer Graphics," *FJCC*, Thompson Books, Washington, D.C., pp. 227-231, 1968.
- [Calvert 78] Calvert, T.W. and J. Chapman, "Notation of Movement with Computer Assistance," *Proceedings of the ACM Annual Conference* 2, pp. 731-736, 1978.
- [Calvert 80] Calvert, T.W., J. Chapman, and A. Patla, "The Integration of Subjective and Objective Data in the Animation of Human Movement," *Computer Graphics (Proceedings Siggraph '80)* 14(3), pp. 198-203, July 1980.
- [Calvert 82a] Calvert, T.W., J. Chapman, and A. Patla, "Specifying and Controlling Human Movement Sequences: The Kinematic Simulation of Human Movement," *IEEE Computer Graphics and Applications*, 1982. (to appear in November)

- [Calvert 82b] Calvert, T.W., J. Chapman, and A. Patla, "The Simulation of Human Movement," *Proceedings Graphics Interface '82*, NCGA, Toronto, pp. 227-234, 1982.
- [Calvert 82c] Calvert, T.W., A. Patla, and J. Chapman, "The Clinical Use of a Computer Assisted Movement Notation System," *Physiotherapy Canada*, 1982. (accepted for publication)
- [Cappozzo 76] Cappozzo, A., F. Figura, M. Marchetti, and A. Pedotti, "The Interplay of Muscular and External Forces in Human Ambulation," *Journal of Biomechanics* 9(1), pp. 35-43, 1976.
- [Cargill 81] Cargill, T.A., "Full-Screen Editing in a Hostile Environment," *Software Practice & Experience* 11(9), pp. 975-981, September 1981.
- [Causley 67] Causley, M., *An Introduction to Benesh Movement Notation*, Man Parrish, London, 1967.
- [Cotton 68] Cotton, A.V., "Notation," *The Dancing Times*, pp. 636, September 1968.
- [Curl 67] Curl, G.F., "An Enquiry into Movement Notation (Part I)," *Laban and Benesh Movement Notations*, Chelsea College of Physical Education, 1967.
- [Davis 72] Davis, M., *Understanding Body Movement: An Annotated Bibliography*, Arno Press, New York, 1972.
- [Davis 75] Davis, M., *Towards Understanding the Intrinsic in Body Movement*, Arno Press, New York, 1975.
- [Dell 70] Dell, C., *A Primer for Movement Description*, Dance Notation Bureau, New York, 1970.
- [Dell 77] Dell, C. and A. Crow, *Space Harmony*, Dance Notation Bureau, New York, 1977. (revised by Irmgard Bartenieff)
- [Dreyfuss 59] Dreyfuss, H., *The Measure of Man: Human Factors in Design*, Whitney Library of Design, New York, 1959.
- [Dreyfuss 72] Dreyfuss, H., *Symbol Sourcebook*, McGraw-Hill Book Company, New York, 1972.
- [Embley 81] Embley, D.W. and G. Nagy, "Behavioral Aspects of Text Editors," *Computing Surveys* 13(1), pp. 33-70, March 1981.
- [Encyc. Brit. 74] *Encyclopaedia Britannica*, Helen Hemingtonway Benton, Chicago, 1974.
- [Eshkol 58] Eshkol, N. and A. Wachman, *Movement Notation*, Weidenfeld and Nicholson, London, 1958.
- [Eshkol 70] Eshkol, N., P. Melvin, H.V. Foerster, and A. Wachman, "Notation of Movement," Report BCL 10.0, Department of Electrical Engineering, University of Illinois, Urbana, Illinois, 1970.
- [Eshkol 75] Eshkol, N., *Right Angled Curves*, Movement Notation Society, 1975. (for the Research Centre for Movement Notation, Faculty of Arts, Tel Aviv University, Israel)
- [Fedak 78] Fedak, J. F., "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, January 1978.
- [Fetter 74] Fetter, W., "A Human Figure Computer Graphics Development for Multiple Applications," *Proceedings Eurocomp Congress, Online, Brunel, England*, pp. 476-488, May 1974.

- [Finseth 80] Finseth, C. A., "Theory and Practice of Text Editors or A Cookbook for An Emacs," MIT/LCS/TM-165, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, May 1980.
- [Finseth 82] Finseth, C.A., "Managing Words: What Capabilities should You have with a Text Editor?," *Byte* 7(4), pp. 302-310, April 1982.
- [Foley 74] Foley, J.D. and V.L. Wallace, "The Art of Natural Graphic Man-Machine Conversation," *Proceedings of the IEEE*, April 1974.
- [Foley 82a] Foley, J.D. and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publications, Massachusetts, 1982.
- [Foley 82b] Foley, J.D., "The Design and Implementation of User-Interface Interfaces," *Siggraph '82 Tutorial No. 7*, ACM, July 1982.
- [Fraser 80] Fraser, C.W., "Syntax-Directed Editing of General Data Structures," TR 80-16, Department of Computer Science, The University of Arizona, Tucson, Arizona 85721, June 1980.
- [Gardner 80] Gardner, J.A., "The FRED Text Editor: Reference Manual," Computer Science Department, University of Waterloo, 1980. (Online documentation)
- [Grieve 76] Grieve, D.W., "Electromyography," pp. 109-149 in *Techniques for the Analysis of Human Movement*, ed. D.W. Grieve, Princeton Book Company, Princeton, N.J., 1976.
- [Goldberg 76] Goldberg, R., *Performance: The Art of Notation*, Studio International, pp. 54-58, July/August, 1976.
- [Gombrich 72] Gombrich, E.H., "The Visual Image," *Scientific American* 227(3), pp. 82-96, September 1972.
- [Gorsky 77] Gorsky, A., *Two Essays on Stepanov Dance Notation*, Cord Special Publication, 1977. (Translated from Russian by Roland John Wiley)
- [Grater 82] Grater, A., *private correspondence*, Institute of Choreology, London, England, 1982.
- [Green 68] Green, F., "Benesh Movement Notation," *The Dancing Times*, pp. 479-483, June 1968.
- [Hackathorn 77] Hackathorn, R.J., "ANIMA II: A 3-D Color Animation System," *Computer Graphics (Siggraph '77)* 11(2), pp. 54-64, Summer 1977.
- [Hall 69] Hall, E.T., *The Hidden Dimension*, Anchor Press, New York, 1969.
- [Hall 64] Hall, F., "Dance Notation and Choreology," *British Journal of Aesthetics* 4(1), pp. 58-66, January 1964.
- [Hall 65] Hall, F., "An Alphabet of Movement," *New Scientist* 28(467), pp. 285-288, October 28, 1965.
- [Hall 66] Hall, F., "Benesh Movement Notation and Choreology," *Dance Scope*, New York, Fall 1966.
- [Hall 67a] Hall, F., "Benesh Notation and Ethnochoreology," *Ethnomusicology* 11(2), pp. 188-198, May 1967.
- [Hall 67b] Hall, F., "Benesh Movement Notation Today," *Ballet Today*, February 1967.
- [Hanavan 66] Hanavan, E.P., "A Personalized Mathematical Model of the Human Body," *Journal of Spacecrafts and Rockets* 3(3), pp. 446-448, March 19, 1966.

- [Hansen 71] Hansen, W., "User Engineering Principles for Interactive Systems," *Proceedings 1971 Fall Joint Computer Conference*, pp. 523-532, 1971.
- [Heck 65] Heck, C.V., I.E. Hendryson, and C.R. Rowe, *Joint Motion - Method of Measuring and Recording*, American Academy of Orthopedic Surgeons, 1965.
- [Herbison-Evans 74] Herbison-Evans, D., "Animated Cartoons by Computer using Ellipsoids," *Proceedings 6th Australian Computer Conference*, pp. 811-823, 1974.
- [Herbison-Evans 78] Herbison-Evans, D., "NUDES 2: A Numeric Utility Displaying Ellipsoid Solids, version 2," *Computer Graphics* 12(3), pp. 354-356, Aug. 1978.
- [Herbison-Evans 79a] Herbison-Evans, D., "A Human Movement Language for Computer Animation," pp. 117-128 in *Language Design and Programming Methodology*, ed. Jeffrey M. Tobias, September 1979. (Proceedings of the Symposium on Language Design and Programming Methodology, Sydney, Australia)
- [Herbison-Evans 79b] Herbison-Evans, D., "Algorithms for Real Time Animation of Drawings of the Human Figure with Hidden Lines Omitted," Technical Report 148, Basser Department of Computer Science, Sydney University, 1979.
- [Hoff 77] Hoff, F., "Dance Preserved at Motsuji," *Dance Research Journal* 9(2), pp. 1-4, 1977.
- [Hornbuckle 67] Hornbuckle, G.D., "The Computer Graphics/User Interface," *IEEE Trans. HFE* 8(1), pp. 17-20, March 1967.
- [Huggins 69] Huggins, W.H. and D.R. Entwisle, "Iconic Communications," *IEEE Transactions on Education* 14(4), pp. 158-163, 1969.
- [Huggins 74] Huggins, W.H. and D.R. Entwisle, *Iconic Communication: An Annotated Bibliography*, John Hopkins Press, Baltimore, 1974.
- [Hutchinson 67] Hutchinson, A., "A Survey of Systems of Dance Notation (Part II)," *Laban Art of Movement Guild Magazine* 20, pp. 25-38, May 1967.
- [Hutchinson 64] Hutchinson, A., "Notation: A Means of International Communication in Movement and Dance," *Impulse*, pp. 82-83, 1963/64.
- [Hutchinson 68a] Hutchinson, A., "Experiences of Dance Notations," *The Dancing Times*, pp. 308-304, March 1968.
- [Hutchinson 68b] Hutchinson, A., "Some Questions about Accuracy," *The Dancing Times*, pp. 425-429, May 1968.
- [Hutchinson 68c] Hutchinson, A., "A Look at the Benesh System," *The Dancing Times*, pp. 364-365, April 1968.
- [Hutchinson 70] Hutchinson, A., "Dance Notation: A Controversy," *Dance Scope* 5(1), pp. 39-55, Fall 1970.
- [Hutchinson 76] Hutchinson, A., *Chronological List of Dance Notation Systems*, 1976. (unpublished work)
- [Hutchinson 77] Hutchinson, A., *Labanotation*, New York: Theatre Arts, 1977. (3rd Edition)
- [IBM 70] "IBM Virtual Machine/System Product: System Product Editor Command and Macro Reference," Program Number 5664-167, IBM, 1970.
- [Jay 57] Jay, L., "A Stick-Man Notation," *Dance Observer*, pp. 7-8, January 1957.
- [Johnson 70] Johnson, A.R., "Dialogue and the Exploration of Context: Properties of An Adequate Interface," presented at *Fourth Annual International Symposium of the American Society of Cybernetics*, Washington, D.C., 1970.

- [Jong 82] Jong, S., "Designing a Text Editor? The User Comes First," *Byte* 7(4), pp. 284-300, April 1982.
- [Joy 80] Joy, W., "An Introduction to Display Editing with Vi," *UNIX Programmer's Manual, Seventh Edition (VAX II Version) 2c*, Computer Science Division, Department of Electrical Engineering and Computer Science, University of Berkeley, September 16, 1980.
- [Kaufman 79] Kaufman, B., "The Simulation of Human Locomotion," MSE Thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, June 1979.
- [Kember 76] Kember, P.A., "The Benesh Movement Notation Used to Study Sitting Behaviour," *Applied Ergonomics* 7(3), pp. 133-136, 1976.
- [Kernighan 78a] Kernighan, B.W., "A Tutorial Introduction to the UNIX Text Editor," *UNIX Programmer's Manual, Version 7*, Bell Laboratories, Murray Hill, New Jersey, September 1978.
- [Kernighan 78b] Kernighan, B.W., "Advanced Editing on UNIX," *UNIX Programmer's Manual, Version 7*, Bell Laboratories, Murray Hill, New Jersey, August 1978.
- [Kitching 76] Kitching, A. and C. Emmett, "The Antics Computer System," *Computer Graphics* 10(4), pp. 13-17, Winter 1976.
- [Kochanek 82] Kochanek, D.H.U., "A Computer System for Smooth Key Frame Animation," M.Math. Thesis, Department of Computer Science, University of Waterloo, 1982.
- [Kolers 69] Kolers, P.A., "Some Formal Characteristics of Pictograms," *American Scientist* 57(3), pp. 348-363, 1969.
- [Laban 66] Laban, R., *Choreutics*, Macdonald & Evans, London, 1966.
- [Laban 75] Laban, R., *Laban's Principles of Dance and Movement Notation*, Macdonald & Evans Ltd., London, 1975. (2nd Edition)
- [Lansdown 77] Lansdown, J., "Computer Choreography and Video," pp. 241-252 in *Computing in the Humanities, Proceedings of the Third International Conference on Computing in the Humanities*, eds. Serge Lusignan and John S. North, The University of Waterloo Press, Waterloo, Canada, August 1977.
- [Lansdown 78] Lansdown, J., "The Computer in Choreography," *IEEE Computer* 11(8), pp. 19-30, August 1978.
- [Ledgard 80] Ledgard, H.F., J. A. Whiteside, W. Seymour, and A. Singer, "An Experiment on Human Engineering on Interactive Software," *IEEE Trans. on Software Engineering* SE-6(6), November 1980.
- [MacKay 82a] MacKay, S.A., R.E. Sayre, and M.J. Potel, "3D Galatea: Entry of Three-Dimensional Moving Points from Multiple Perspective Views," *Computer Graphics (Siggraph '82)* 16(3), July 1982.
- [MacKay 82b] MacKay, S.A., "Techniques for Frame Buffer Animation," M.Math. essay, Department of Computer Science, University of Waterloo, 1982.
- [Martin 73] Martin, J., *Design of Man-Machine Dialogues*, Englewood Cliffs, N.J., 1973.
- [Massine 76] Massine, L., *Massine on Choreography*, Faber & Faber, London, 1976.



- [McGuinness-Scott 82] McGuinness-Scott, J., *Benesh Movement Notation: An Introduction to Recording Clinical Data*, Institute of Choreology, London, England, 1982.
- [McLaughlin 76] McLaughlin, P., "Balanchine Forever," *Pennsylvania Gazette*, pp. 25-33, October 1976.
- [McNair 79] McNair, B., "A Language for Notating Human Movement," M.Sc. Thesis, Basser Department of Computer Science, Sydney University, 1979.
- [McNair 80] McNair, B., D. Herbison-Evans, and N. Neilands, "Computer Assisted Choreography Teaching," *Proceedings of the 11th Annual Australian Colleges of Advanced Education*, pp. 282-287, May 1980.
- [Mead 68] Mead, M. and R. Modley, "Communication Among All People, Everywhere," *Natural History* 77(7), pp. 56-63, 1968.
- [Mendo 75] Mendo, G. and L. B. Archer, "Computer-Choreology Project I - Specifications for the Character Recognition and Syntax Analysis of Benesh Movement Notation," in *A Study of Computer Choreography*, Institute of Choreology, London, 1975.
- [Menosky 82] Menosky, J., "Video Graphics & Grand Jetes," *Science*, pp. 25-32, May 1982.
- [Mertens 81] Mertens, S., "Taking Note of Dance," *Vancouver Sun*, September 28, 1981.
- [Michlin 75] Michlin, J., M.E. Figliuzzi, and B.R. Fowler, "A User's Guide to TSO-QED," Bell Laboratories, Feb. 1975.
- [Miles 76] Miles, A., *Labanotation for Ballet Dancers*, Dance Notation Bureau, New York, 1976.
- [Miller 56] Miller, G., "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychology Review* 63(2), pp. 81-97, March 1956.
- [Mills 81] Mills, M.I., *Telidon Behavioural Research 3: A Study of the Human Response to Pictorial Representations on Telidon*, Department of Communications, Ottawa, 1981.
- [Mills 82] Mills, M.I., "Cognitive Schemata and the Design of Graphics Displays," *Proceedings Graphics Interface '82*, NCGA, Toronto, pp. 3-12, 1982.
- [Modley 47] Modley, R., *Pictograms and Graphs*, Harper and Brothers, 1947.
- [Moran 81] Moran, T.P., "An Applied Psychology of the User," *Computing Surveys* 13(1), pp. 1-11, March 1981.
- [Morrice 67] Morrice, N., "Advantages of Benesh Movement Notation to a Choreographer," *Ballet Today*, January-February 1967.
- [Morris 28] Morris, M., *The Notation of Movement*, Kegan Paul, Trench, Trubner & Co. Ltd., London, 1928.
- [Mossford 67] Mossford, L., "Advantages of Benesh Movement Notation to a Ballet Company," *Ballet Today*, January-February 1967.
- [Newman 79] Newman, W.M. and R.F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill Book Company, 1979. (2nd edition)
- [Nickerson 81] Nickerson, R.S., "Why Interactive Computer Systems are Sometimes not Used by People Who might Benefit from Them," *Int. J. Man-Machine Studies* 15(4), pp. 469-483, 1981.

- [Noll 67] Noll, A.M., "Choreography and Computers," *Dance Magazine*, pp. 43-46 & 81-82, January 1967.
- [O'Rourke 78] O'Rourke, J., "Three-Dimensional Motion of a Three Link System," Technical Report, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, June 1978.
- [Officer 77] Officer, J.M., "Computerization of Human Movement - A Tool for Analysis," *Proc: The International Congress of Physical Activity Sciences*, Quebec City, 1977.
- [Parent 77] Parent, R.E., "A System for Sculpting 3-D Data," *Computer Graphics* 11(2), pp. 138-147, Summer 1977.
- [Parker 82] Parker, M., *Benesh Movement Notation Encyclopaedia*, Institute of Choreology, London, England, 1982.
- [Pettis 78] Pettis, K., "The Screen Editor K," Computer Science Department, University of Arizona, 1978. (unpublished manuscript)
- [Pierrynowski 80] Pierrynowski, M.R., "Three-Dimensional Filming Using the Direct Linear Transformation," Department of Kinesiology, Simon Fraser University, 1980. (Unpublished work)
- [Pierrynowski 82] Pierrynowski, M.R., "A Physiological Model for the Solution of Individual Muscle Forces during Normal Human Walking," Ph.D. Thesis, Department of Kinesiology, Simon Fraser University, July 1982.
- [Potter 75a] Potter, T.E., "Three-Dimensional Human Display Model for Two-Dimensional Computer Graphics," Report MIE-006, Department of Mechanical and Industrial Engineering, Clarkson College of, Technology, Potsdam, N. Y., 1975.
- [Potter 75b] Potter, T.E. and K.D. Willmert, "Three-Dimensional Human Display Model," *Computer Graphics* 9(1), pp. 102-110, Spring 1975.
- [Preston-Dunlop 69a] Preston-Dunlop, V., "A Notation System for Recording Observable Motion," *International Journal of Man-Machine Studies* 1(4), pp. 361-386, 1969.
- [Preston-Dunlop 69b] Preston-Dunlop, V., *Practical Kinetography Laban*, MacDonald & Evans Ltd., London, 1969.
- [Reeves 81] Reeves W., "Inbetweening for Computer Animation Utilizing Moving Point Constraints," *Computer Graphics (Siggraph'81)* 15(3), pp. 263-269, August 1981.
- [Reichardt 68] Reichardt, J., "Computer Programmed Choreography," in *Cybernetic Serendipity*, ed. J. Reichardt, Studio International, London and New York, 1968.
- [Rouse 81] Rouse, W. B., "Human-Computer Interaction in the Control of Dynamic Systems," *Computing Surveys* 13(1), pp. 71-99, March 1981.
- [Ryman 81] Ryman, R., A. Ryman, and R. Hughes, "The Structure of Benesh Movement Notation," Dance Group, University of Waterloo, Canada, 1981. (unpublished resource material for computer editing project)
- [Ryman 82a] Ryman, R., B. Singh, J.C. Beatty, and K.S. Booth, "A Computerized Editor for Benesh Movement Notation," June 1982. (Presented at the 10th Annual Dance in Canada Conference, Ottawa)

- [Ryman 82b] Ryman, R. and A. Grater, *Benesh Movement Notation: Elementary Syllabus Ballet Application*, Institute of Choreology, London, England, 1982.
- [Savage 77a] Savage, G.J. and J. M. Officer, "CHOREO: An Interactive Computer Model for Dance," *5th Man-Computer Communications Conference*, Calgary, Academic Press Inc. (London) Limited, pp. 233-249, May 1977.
- [Savage 77b] Savage, G.J. and J.M. Officer, "Interactive Computer Graphics Methods for Choreography," in *Computing in the Humanities, Proceedings of the Third International Conference on Computing in the Humanities*, The University of Waterloo Press, Waterloo, Canada., August 1977.
- [Savage 79] Savage, G.J., J.M. Officer, and G. McDougall, "Computer Graphics Simulation of Body Movement Language," *Proceedings 6th Man-Computer Communications Conference*, pp. 209-217, 1979.
- [Sealey 81] Sealey, D., "Computers and Labanotation," *Proceedings of the Twelfth Biennial Conference, Columbus, Ohio*, International Council of Kinetography Laban, pp. 126-127, August 1981.
- [Seybold 81a] Seybold, J., "The Xerox Star: A Professional Workstation," *The Seybold Report on Word Processing* 4(5), May 1981.
- [Seybold 81b] Seybold, J., "Xerox's Star," *The Seybold Report* 10(16), April 27, 1981.
- [Shoup 79] Shoup, R.G., "Colour Table Animation," *Computer Graphics* 13(2), pp. 8-13, August 1979.
- [Smith 82] Smith, D.C., C. Irby, R. Kimball, and B. Verplank, "Designing the Star User Interface," *Byte* 7(4), pp. 242-282, April 1982.
- [Smoliar 77] Smoliar, S.W. and L. Weber, "Using the Computer for a Semantic Representation of Labanotation," pp. 253-261 in *Computing in the Humanities, Proceedings of the Third International Conference on Computing in the Humanities*, eds. Serge Lusignan and John S. North, The University of Waterloo Press, Waterloo, Canada, August 1977.
- [Smoliar 78] Smoliar, S.W., "A Lexical Analysis of Labanotation with an Associated Data Structure," *Proceedings of the ACM Annual Conference*, pp. 727-730, 1978.
- [Smoliar 80] Smoliar, S.W., "Computers Helping Dance Notation Help the Dance: a Vision," *National Computer Conference*, pp. 67-71, 1980.
- [Spegel 75] Spegel, M., "Programming of Mechanism Motion," Technical Report No. CRL-43, Division of Applied Science, New York University, New York, November 1975.
- [Stepanov 59] Stepanov, V.I., *Alphabet of Movements of the Human Body*, The Golden Head Press, Cambridge, 1959.
- [Strauss 77] Strauss, G.B., C. Wing, and L. Yuen-wah, "Translated Excerpts of Chinese Dance Notation," *Dance Research Journal* 9(2), pp. 6-11, 1977.
- [Teitelbaum 80] Teitelbaum, T. and T. Raps, "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment," TR 80-421, Department of Computer Science, Cornell University, Ithaca, New York 14853, May 1980.
- [Tilbrook 76] Tilbrook, D., "A Newspaper Page Layout System," M.Sc. Thesis, Department of Computer Science, University of Toronto, 1976.
- [Townsend 77] Townsend, M.A., M. Izak, and R.W. Jackson, "Total Motion Knee Goniometry," *J. Biomechanics* 10(3), pp. 183-193, 1977.

- [Trucco 81] Trucco, T., "Notating the Dance," *Ballet News* 2(8), pp. 16-18 & 42-43, February 1981.
- [Turnbaugh 70] Turnbaugh, D.B., "Dance Notation: Potential and Problems," *Dance Scope* 4(2), pp. 39-47, Spring 1970.
- [Ubell 76] Ubell, E., "Dance Notation Steps into a New Era," *New York Times, Section 2*, pp. 12-19, 24 October 1976.
- [Walter 81] Walter, T., "How Dance Classics are Preserved," *Saturday Review*, pp. 60-61, November 1981.
- [Weber 78] Weber, L., S.M. Smoliar, and N.I. Badler, "An Architecture for the Simulation of Human Movement," *Proceedings of the ACM Annual Conference*, 2, pp. 737-745, 1978.
- [Willmert 75] Willmert, K.D. and T.E. Potter, "An Improved Human Display Model for Occupant Crash Simulation Programs," *Computers & Graphics* 2(2), Pergamon Press, Great Britain, pp. 51-54, May 1975.
- [Willmert 77] Willmert, K.D., "Occupant Model for Human Motion," *Computers & Graphics* 1(3), Pergamon Press, Great Britain, pp. 123-128, 1977.
- [Withrow 70] Withrow, C., "A Dynamic Model for Computer-Aided Choreography," Technical Report UTEC-CSc-70-103, Department of Computer Science, University of Utah, Salt Lake City, June 1970.
- [Wolofsky 74] Wolofsky, Z., "Computer Interpretation of Selected Labanotation Commands," M.Sc. Thesis, Kinesiology Department, Simon Fraser University, Burnaby, B.C., Canada, 1974.
- [Zeltzer 82] Zeltzer, D., "Representation of Complex Animated Figures," *Proceedings Graphics Interface '82*, NCGA, Toronto, pp. 205-211, 1982.
- [Zorn 05] Zorn, F.A., "Grammar of the Art of Dancing," Franklin, New York, 1905.

## **Appendix A**

List of known movement notation systems in  
chronological order of their development [Hutchinson 76].

<b>Year</b>	<b>Name</b>	<b>Type of System</b>	<b>Country</b>
1588	Arbeau	Letter	France
Late 15 C.	Cervera	Abstract (letters)	Spain
1650	Playford	Words, floor plans	England
1661	Carducci	Words, pictures (horse ballets)	Italy
1671	Beauchamp	Track (unpublished)	France
1700	Feuillet	Track	France
1751	Favier	Abstract music	France
1762	De La Cuisse	Floor plans	France
1768	Landrin	Words, Floor plans	France
1800	Despreaux	Letters	France
1831	Theleur	Abstract	England
1832	Biosca	Floor plans	Spain
1852	Saint-Leon	Stick figure	France
1855	Bournonville	Words/signs	Denmark
1855	Klemm	Music notes	Germany
1859	Adice	Figure drawings	France
1880	Manzotti	Floor plans	Italy
1885	Soret	Line photographs	Switzerland
1887	Zorn	Stick figure	Germany
1892	Stepanov	Music notes	Russia
1892	Poli	Letters/numbers	France
1892	Giraudet, A.	Letters/numbers	France
1911	Zoder	Words (folk)	Austria
1918	Nijinsky	Music notes	Russia
1919	Desmond	Stick figure	Germany
1923	Alexander	Letters/signs	U.S.A.
1926	Grimm-Reiter	Abstract	Germany
1927	Fischer-Klamt	Abstract	Germany
1927	Kool	Stick figure, music, floor plans	Holland
1927	Peters	Diagrams/music	France
1928	Laban	Abstract	Austria
1928	Parnac	Stick figure	France
1928	Morris	Abstract	England
1928	Sotonin	Abstract	Russia
1928	Wailes	Abstract, pictorial	England
1931	Conte	Music notes	France
1931	Meunier	Word abbreviations, signs	France
1932	Chiesa	Music notes	Italy
1934	Cross	Letters, signs, numbers	U.S.A.
1935	Zadra	Abstract	Italy
1939	Babitz	Visual (stick figure)	U.S.A.
1940	Korty	Music, figures, signs	Germany
1940	Ruskaja	Abstract	Italy
1940	Lissitzian	Stick figure	Russia
1940	Schillinger	Abstract	U.S.A.
1942	Craighead	Stick figure	U.S.A.
1945	Nikolais	Music notes	U.S.A.
1946	Saunders	Words	U.S.A.
1949	Humphery	Stick figure	U.S.A.
1946	Ivancan	Music notes	Yugoslavia
1950	Kurath	Abstract	U.S.A.
1950	Tsonev		Bulgaria
1951	Afdnt	Stick figure	Germany
1952	Raisz	Floor plans	U.S.A.

1952	Birdwhistell	Abstract	U.S.A.
1954	Bourgat	Abstract	France
1954	Misslitz	Stick figure	Germany
1955	Loring/Canna	Abstract	U.S.A.
1955	Katzarova	Abstract (folk)	Bulgaria
1955	Benesh	Visual (Stick figure)	England
1956	Proca-Ciortea	Letters/abstract	Romania
1957	Jay	Stick figure	U.S.A.
1958	Eshkol/Wachmann	Abstract/numbers	Israel
1959	Fee	Abstract	U.S.A.
1960	Paige (Apegian)	Abstract	U.S.A.
1960	Varkovitsky	Abstract	Russia
1964	McCraw	Music notes	U.S.A.
1965	Halprin	Floor plans	U.S.A.
1965	Suna	Abstract/figure	Latvia
1968	Blom	Letters/signs	Norway
1969	Vasilescu/Tita	Abstract/music	Romania
1971	Haralampiev	Music notes	Bulgaria
1973	Bakka	Abstract	Norway
1973	Pajtondziev	Music notes	Yugoslavia
1973	Popescu-Judetz	Letters/signs	U.S.A.
1973	Schwale-Brame	Abstract	U.S.A.
1973	Sutton	Stick figure	U.S.A.
1974	Fitz	Abstract	U.S.A.
1979	Jorgensen	Abstract	Denmark

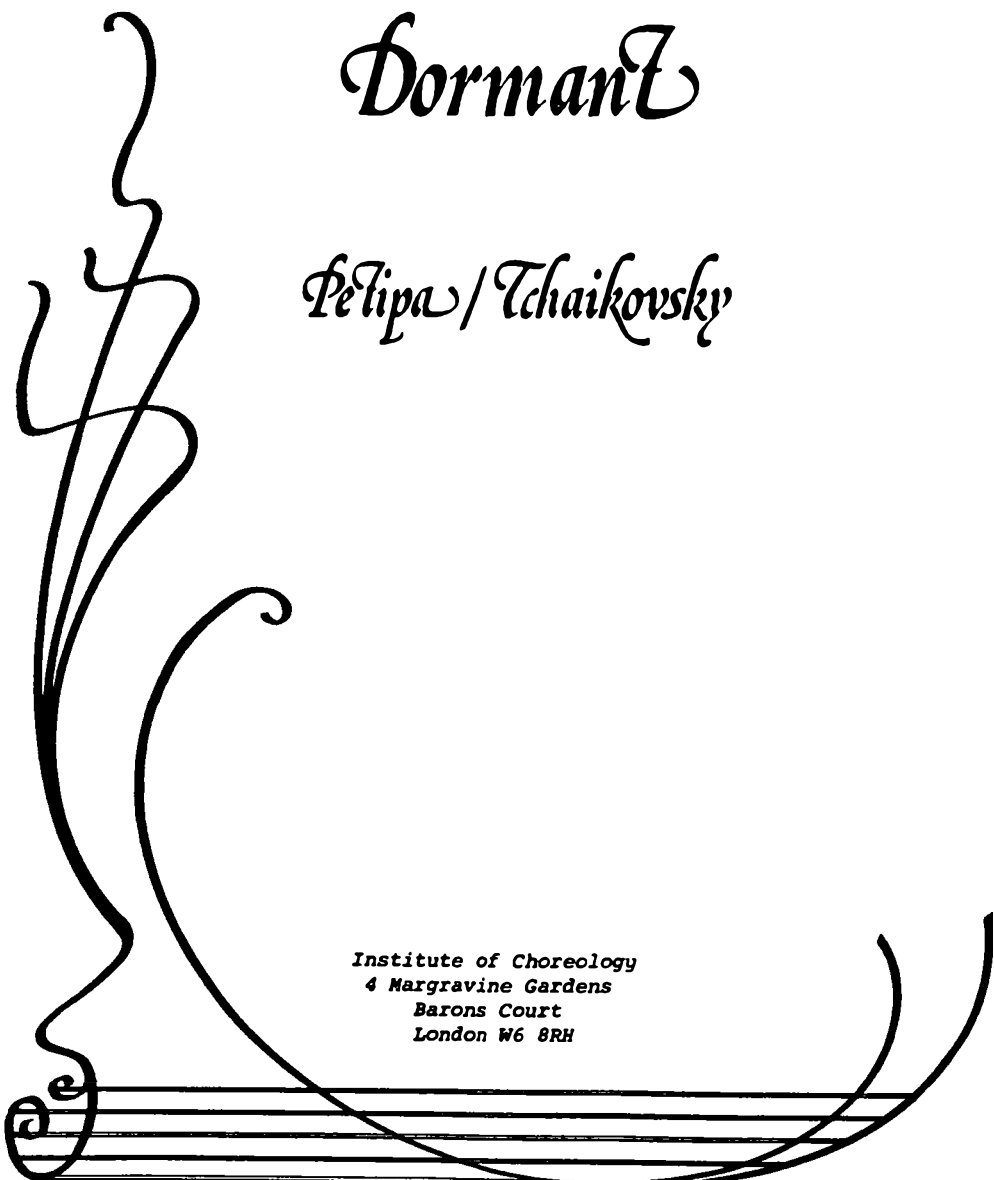
## **Appendix B**

A sample Benesh score.



# *La Belle au Bois Dormant*

*Petipa / Tchaikovsky*



*Institute of Choreology  
4 Margravine Gardens  
Barons Court  
London W6 8RH*

LA BELLE AU BOIS DORMANT ACT III  
AURORA'S VARIATION

as remembered by Mme. Cleo Nordi  
to whom it was taught in Berlin  
by Preobrajenska in 1923.

Choreography: Petipa

Music: Tchaikowsky

LA BELLE AU BOIS DORMANT ACT III  
AURORA'S VARIATION

The image shows seven staves of handwritten musical notation. The notation is dense and includes various symbols such as notes, rests, and ornaments. The first staff begins with a double bar line and a key signature of one flat. The notation is written in a style characteristic of Benesh Movement Notation, with many notes and rests connected by lines and curves. The second staff is mostly empty, with a few notes at the end. The third staff continues the notation with many notes and rests. The fourth staff has a double bar line and a key signature change to one sharp. The fifth staff has a double bar line and a key signature change to one flat. The sixth staff has a double bar line and a key signature change to one sharp. The seventh staff has a double bar line and a key signature change to one flat. The notation is very detailed and includes many small details such as slurs, ties, and ornaments.

Benesh Movement Notation © Rudolf Benesh London 1955.  
The Benesh Institute of Choreology Ltd. © London 1982.

Four empty staves of musical notation, each consisting of five lines. They are arranged vertically and are completely blank.

## **Appendix C**

**Finite state diagrams of the editor,  
showing the state transitions for each command as actually implemented.**

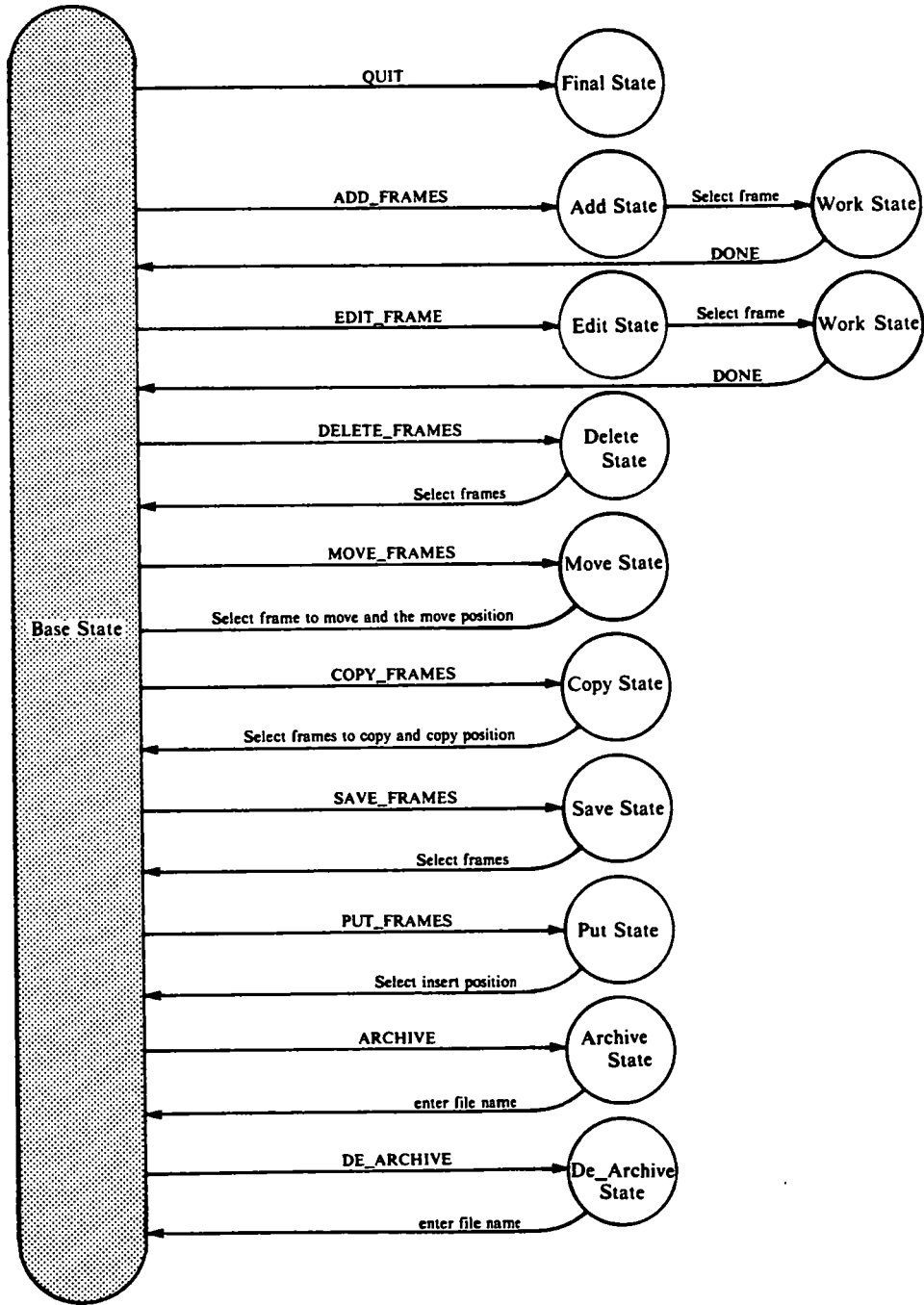


Figure C.1. State diagram of the editor.

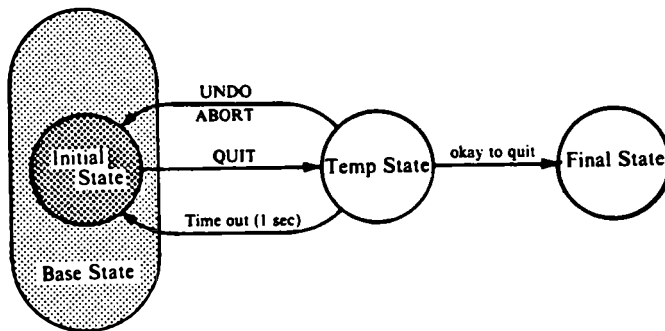


Figure C.2. QUIT command.

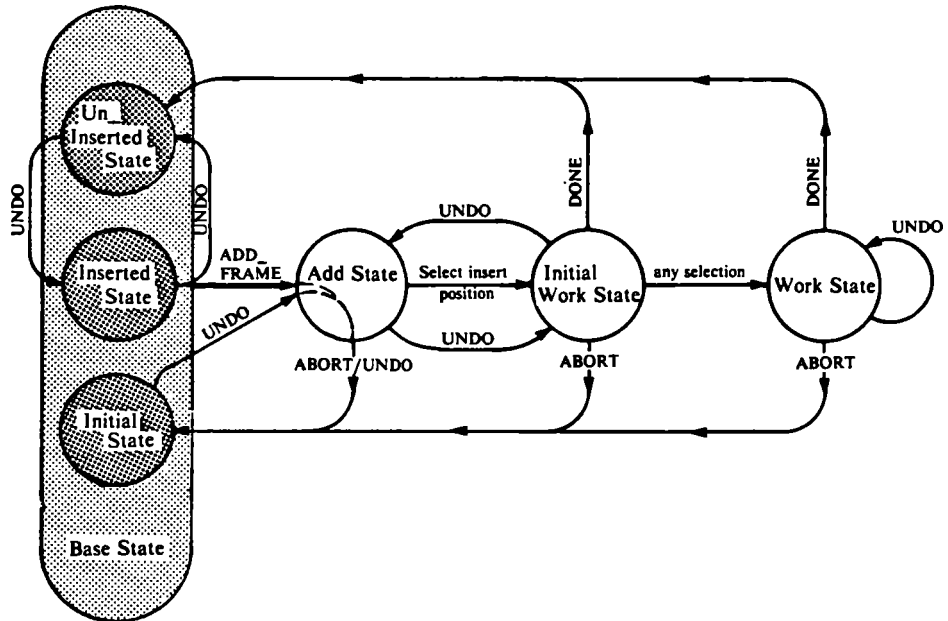


Figure C.3. ADD\_FRAME command.

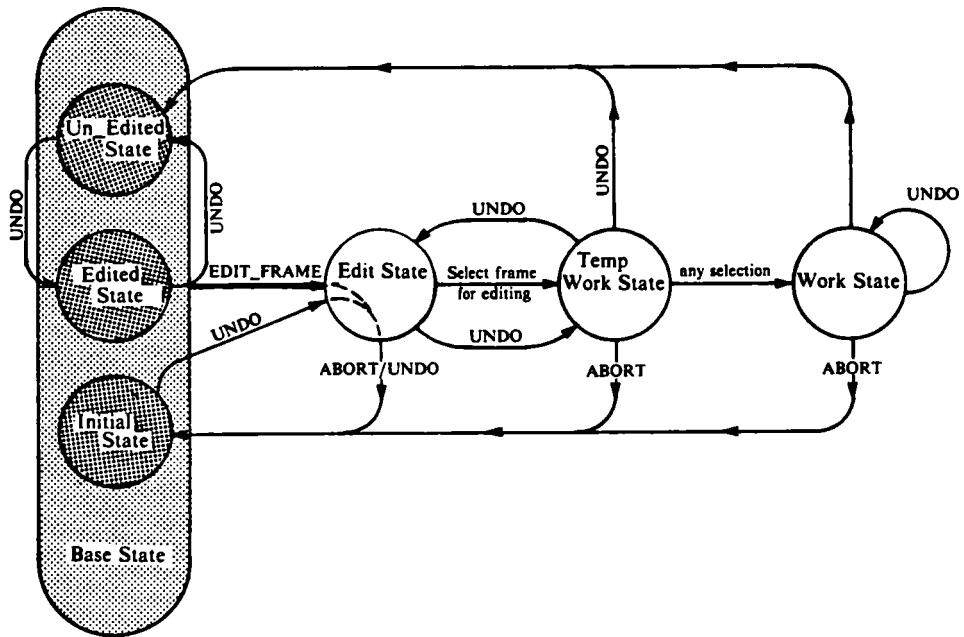


Figure C.4. EDIT frame command.



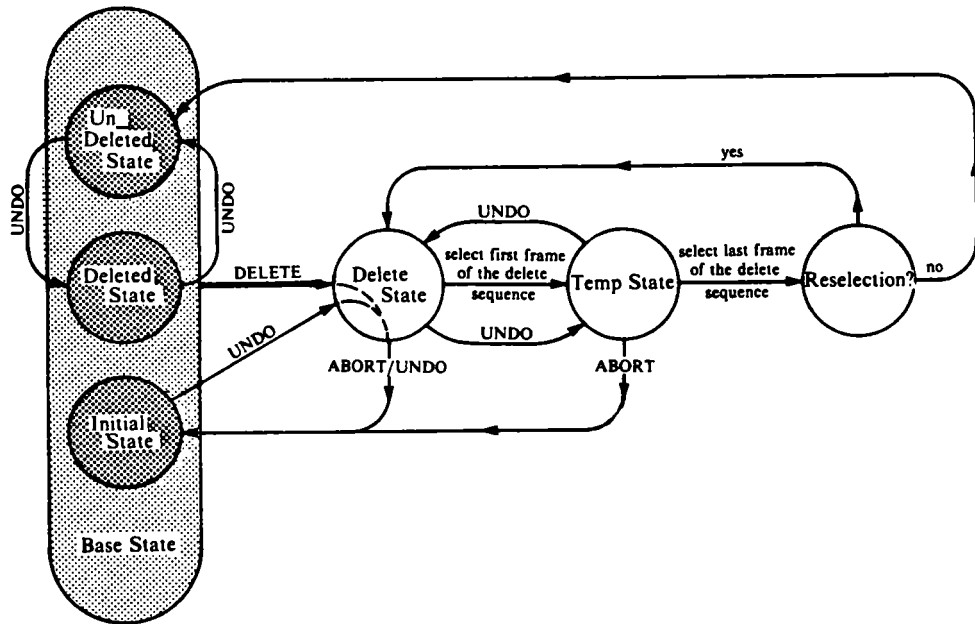


Figure C.5. DELETE frames command.

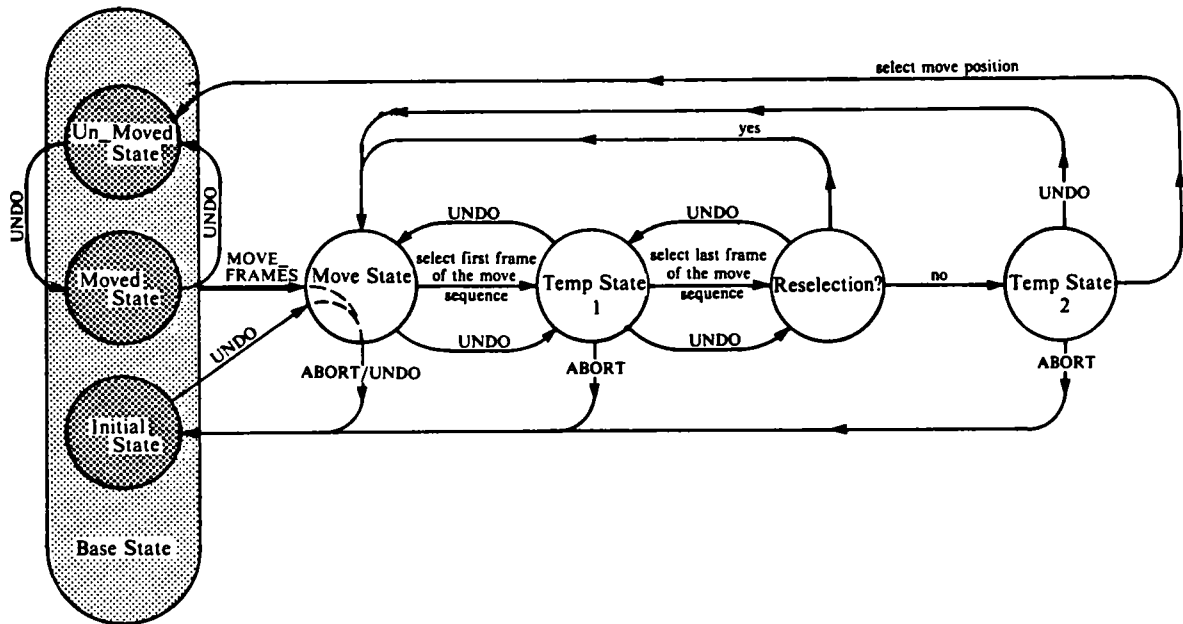


Figure C.6. MOVE frames command.

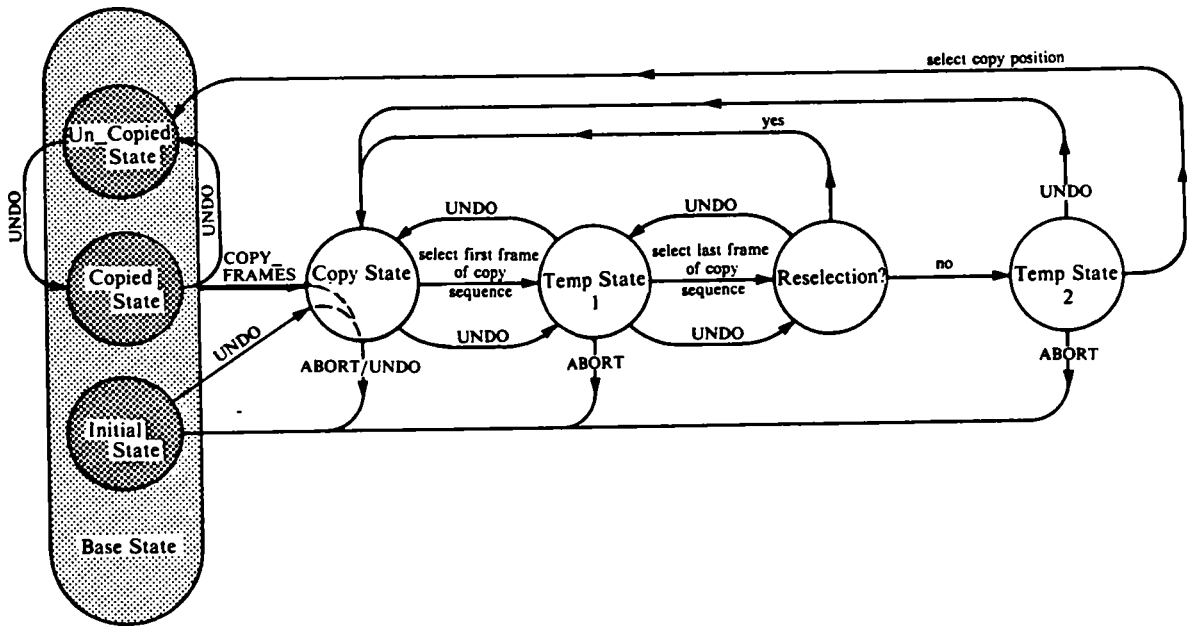


Figure C.7. COPY frames command.

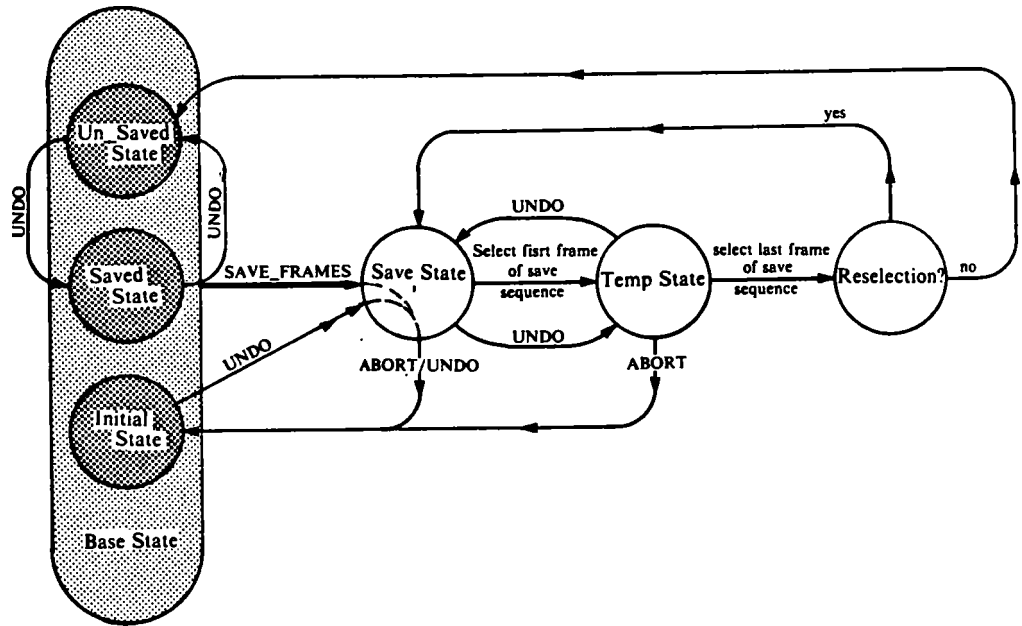


Figure C.8. SAVE frames command.

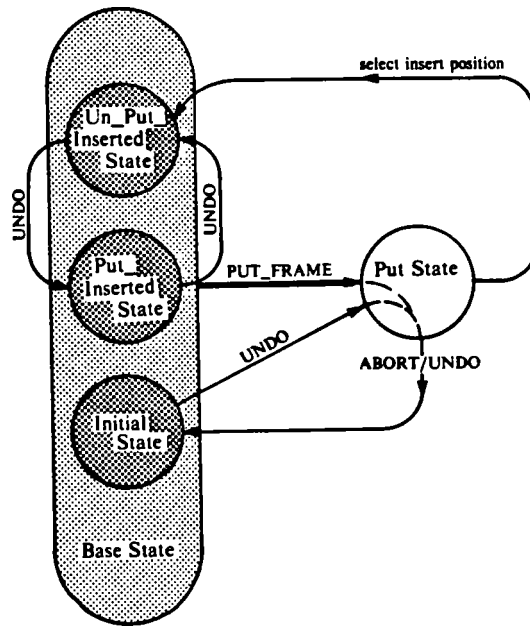


Figure C.9. PUT frames command.

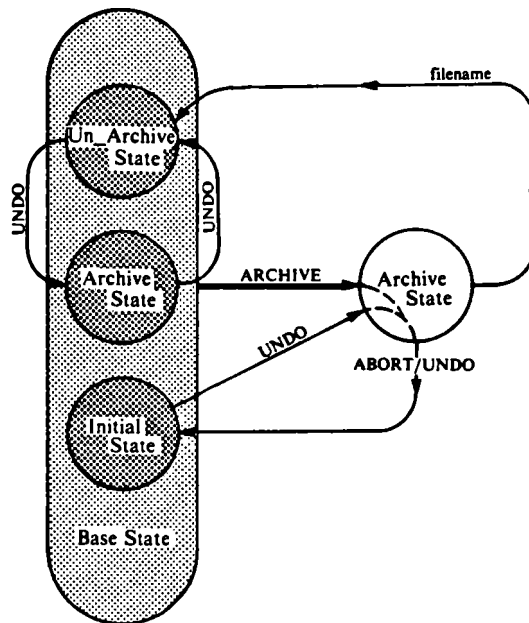


Figure C.10. ARCHIVE score command.

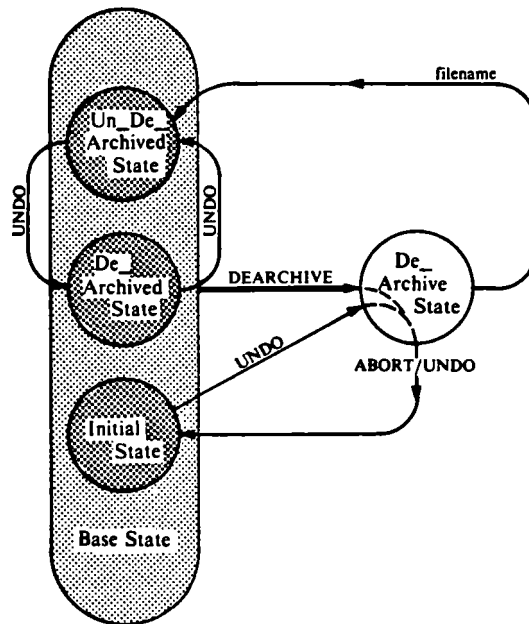
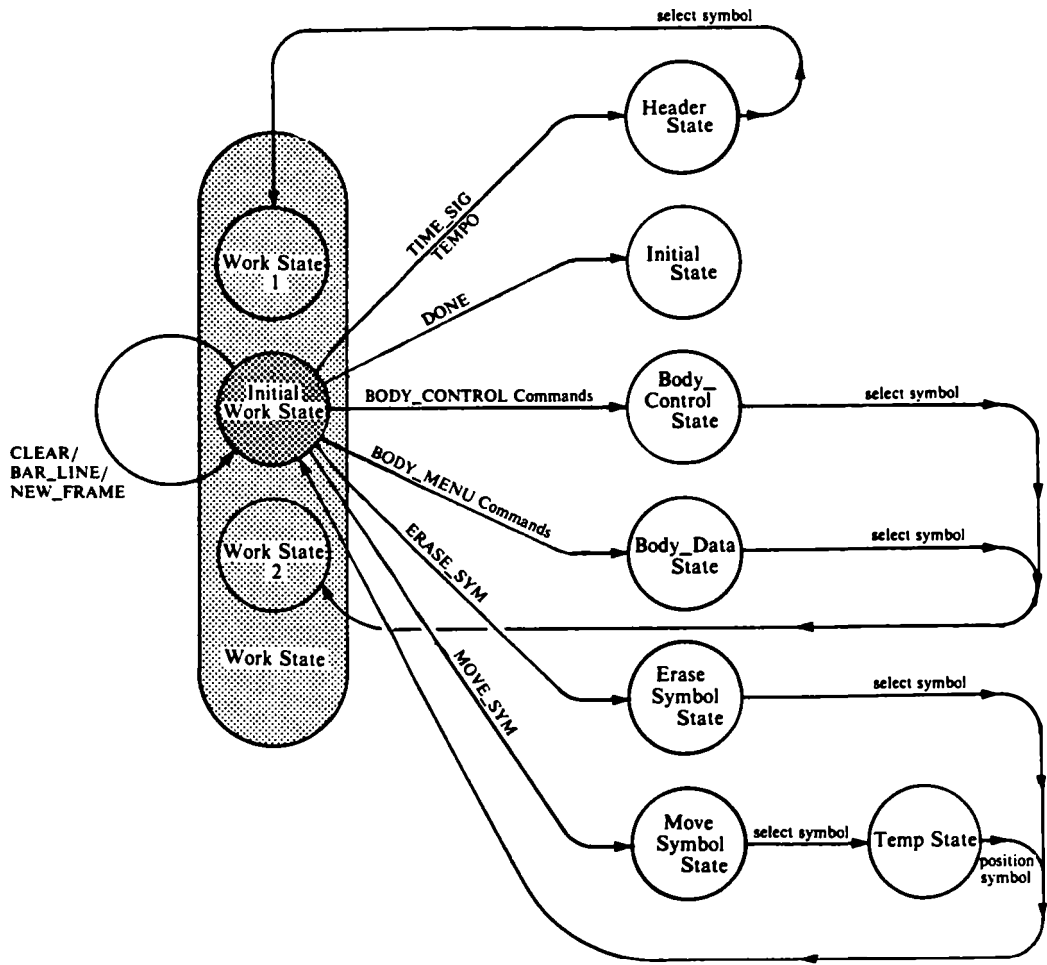


Figure C.11. DE\_ARCHIVE command.



**BODY\_CONTROL commands:**  
EFFORT  
RHYTHM  
DIRECTION  
NOTES

**BODY\_MENU Commands:**  
Left arm  
Right arm  
Left leg  
Right leg  
Head  
Torso  
Pelvis

Figure C.12. Editing a frame.



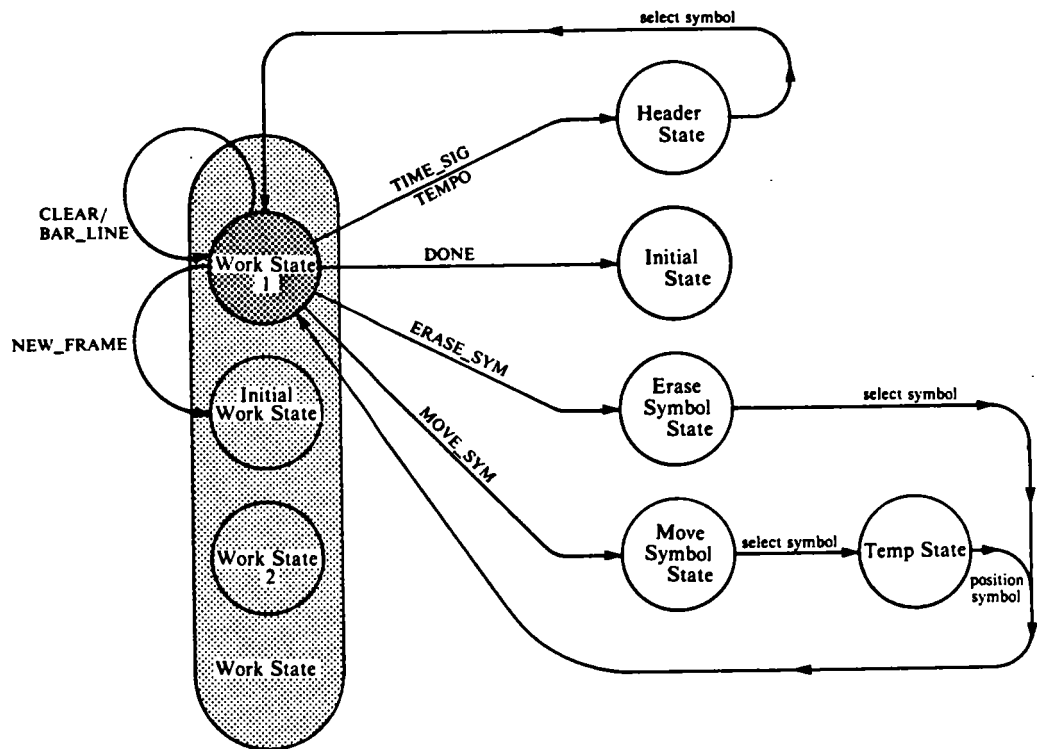


Figure C.12. Editing a frame continued ...

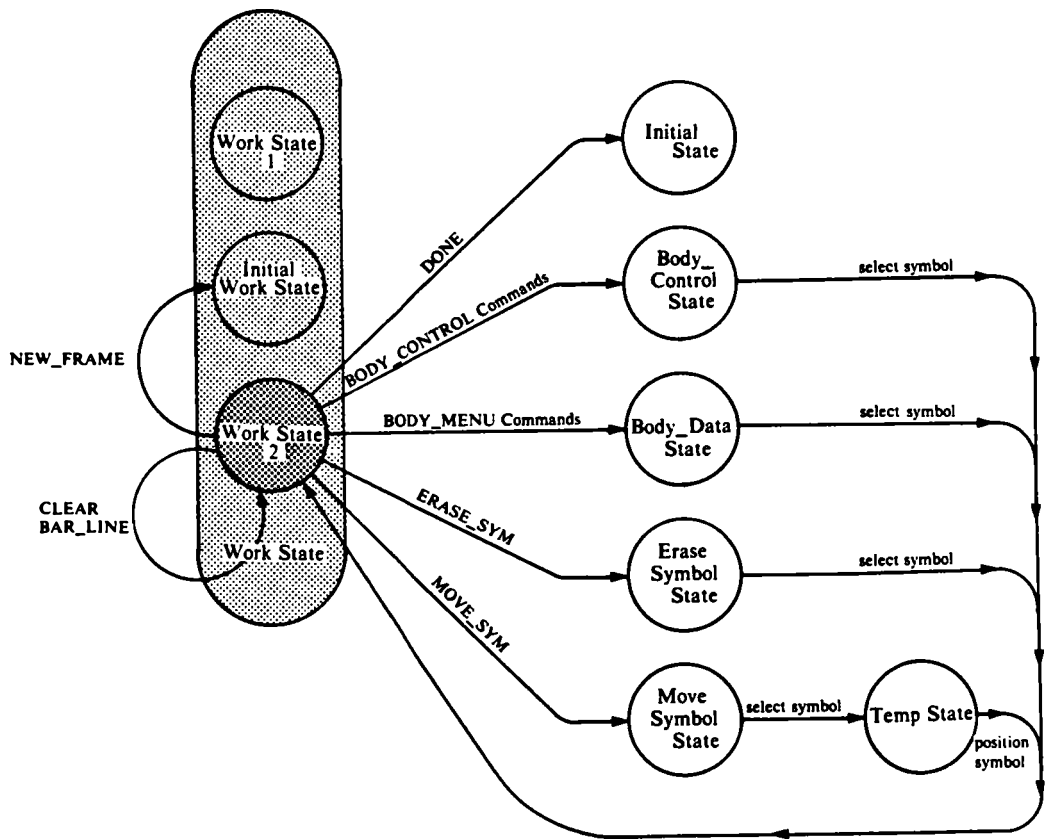


Figure C.12. Editing a frame continued ...

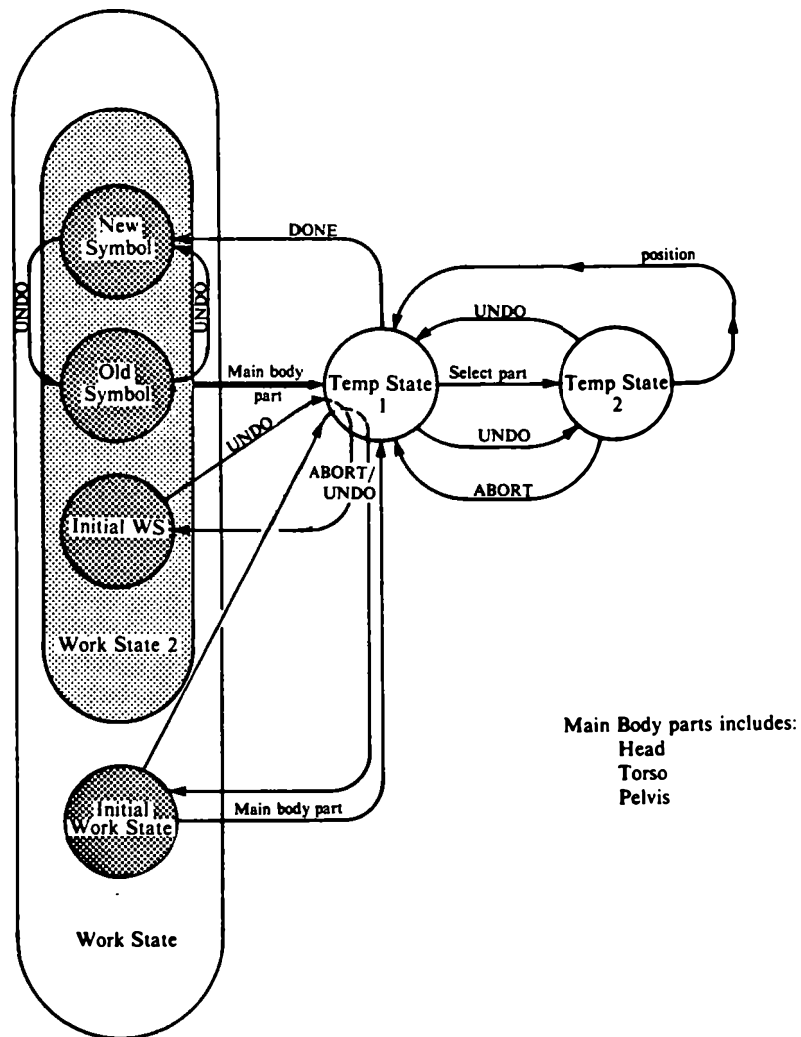


Figure C.13. Main body part positioning.

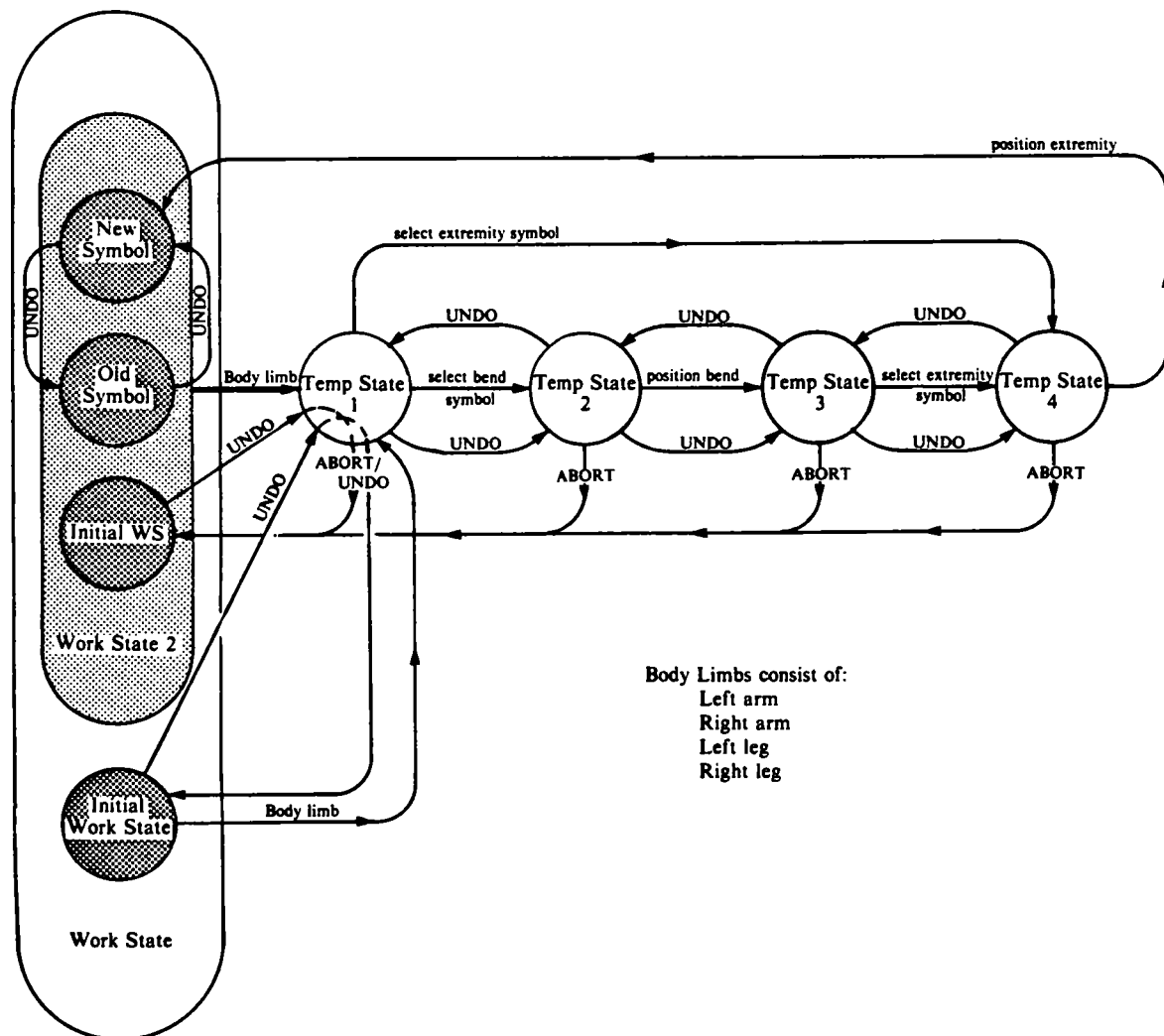


Figure C.14. Body limb positioning.

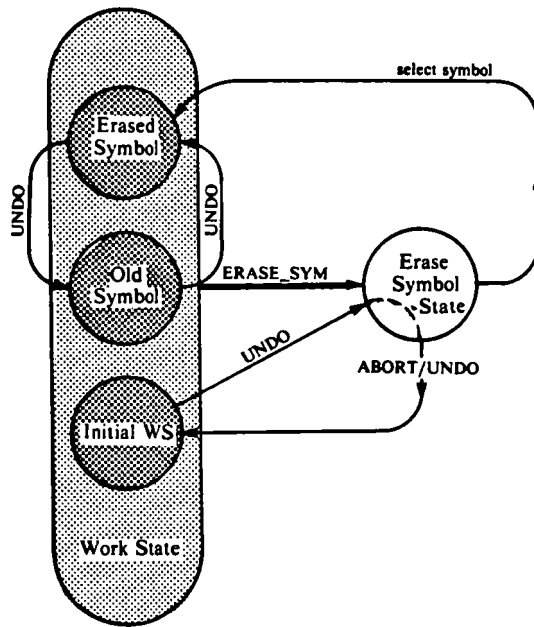


Figure C.15. ERASE\_SYMBOL command.

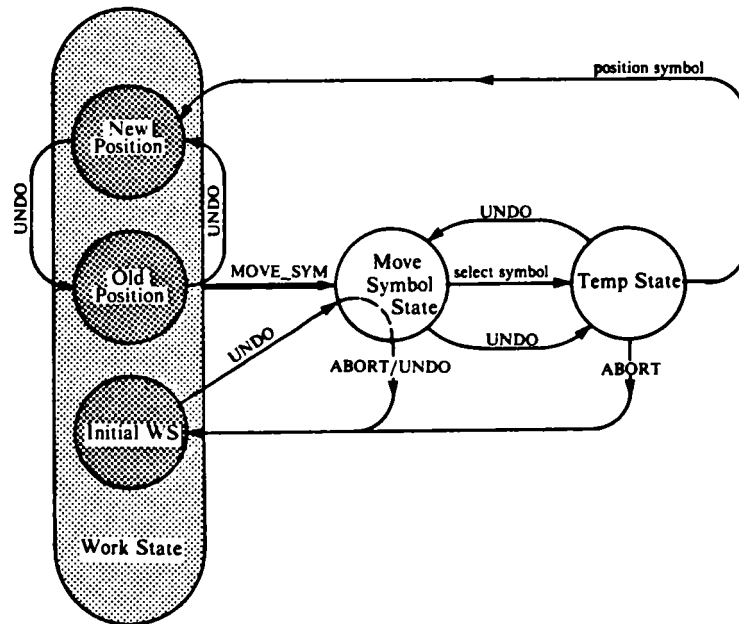


Figure C.16. MOVE\_SYMBOL command.

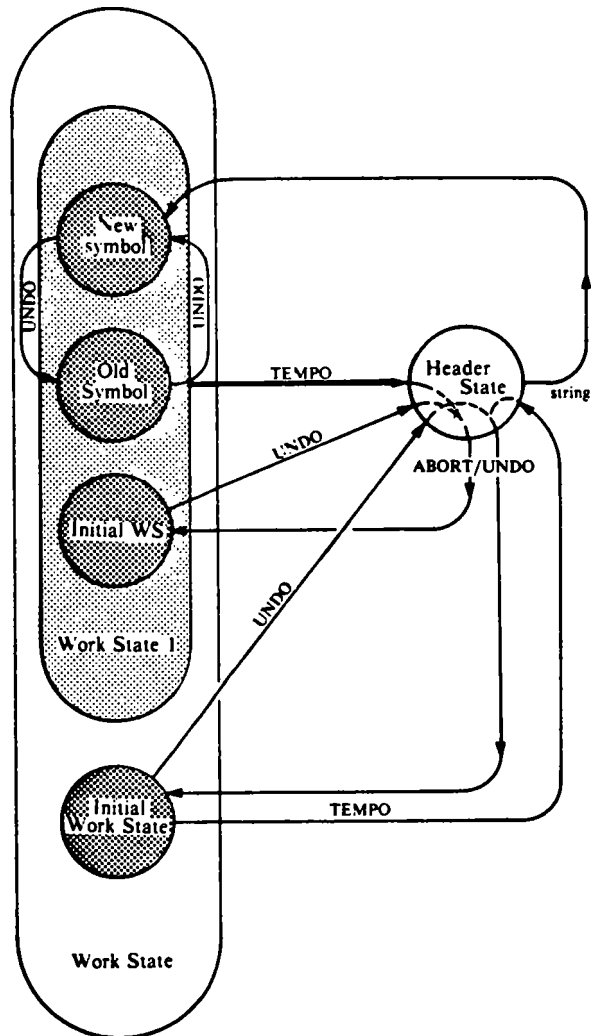


Figure C.17. TEMPO command.

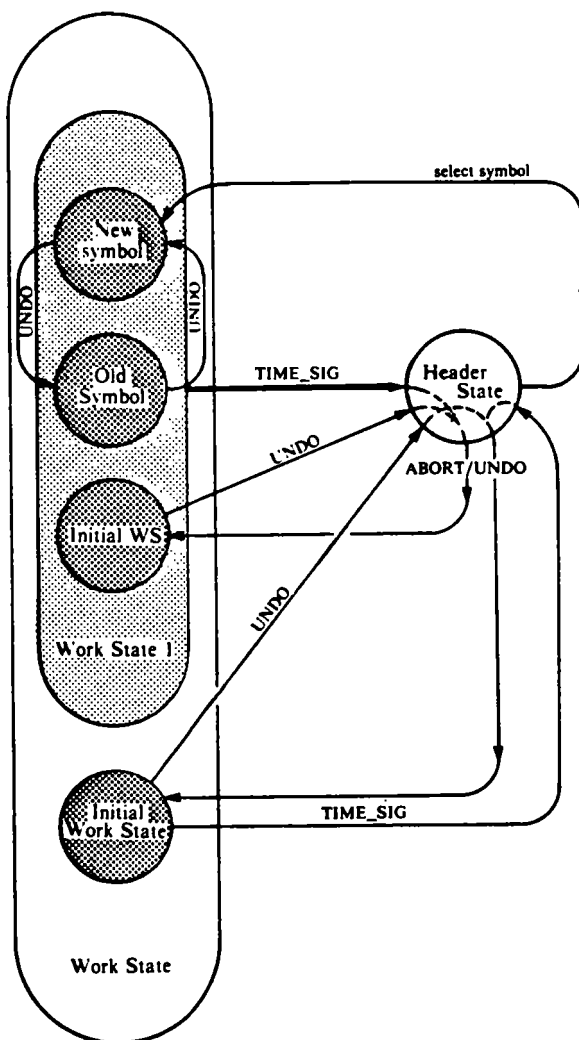


Figure C.18. TIME\_SIGNATURE command.



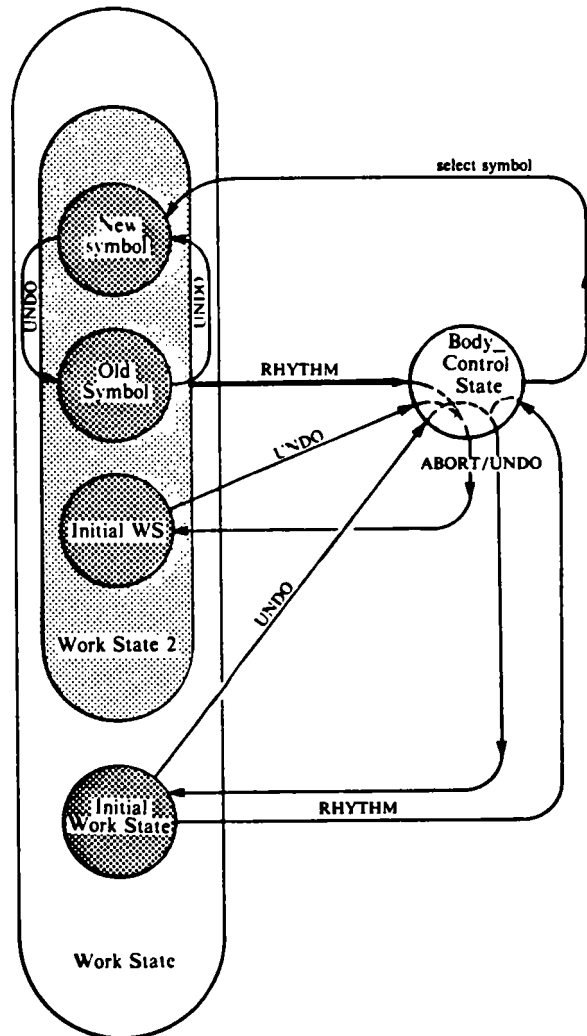


Figure C.19. RHYTHM command.

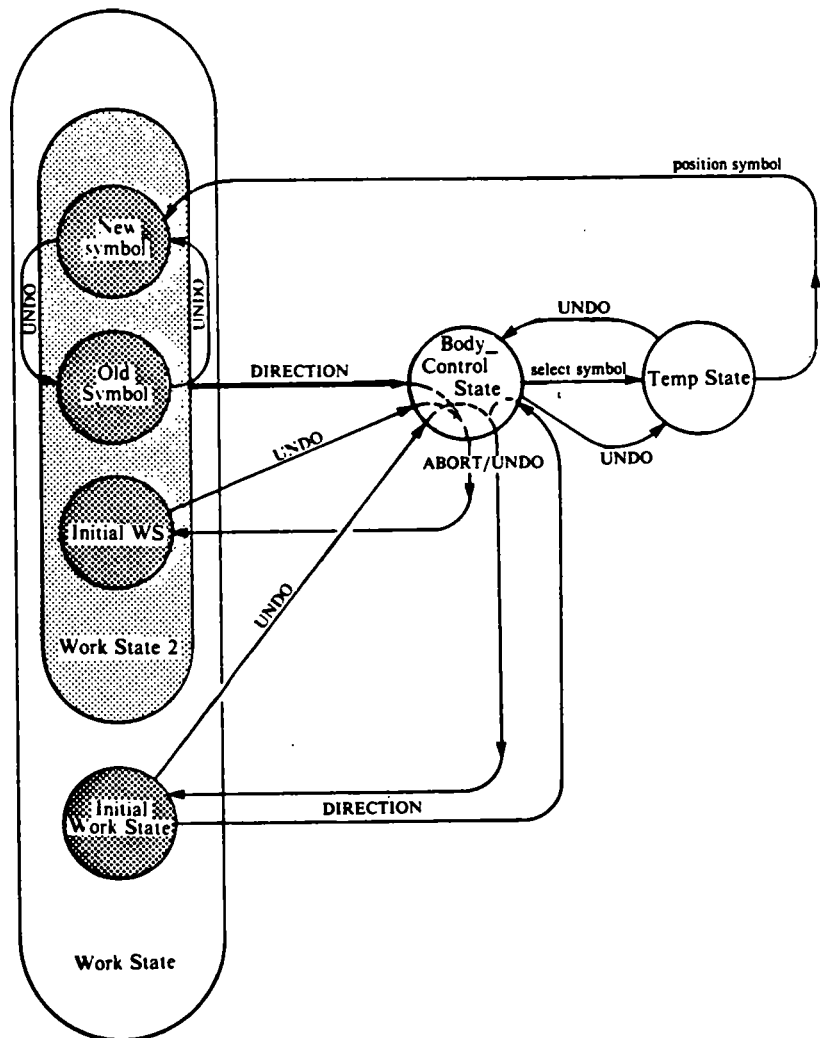


Figure C.20. DIRECTION command.

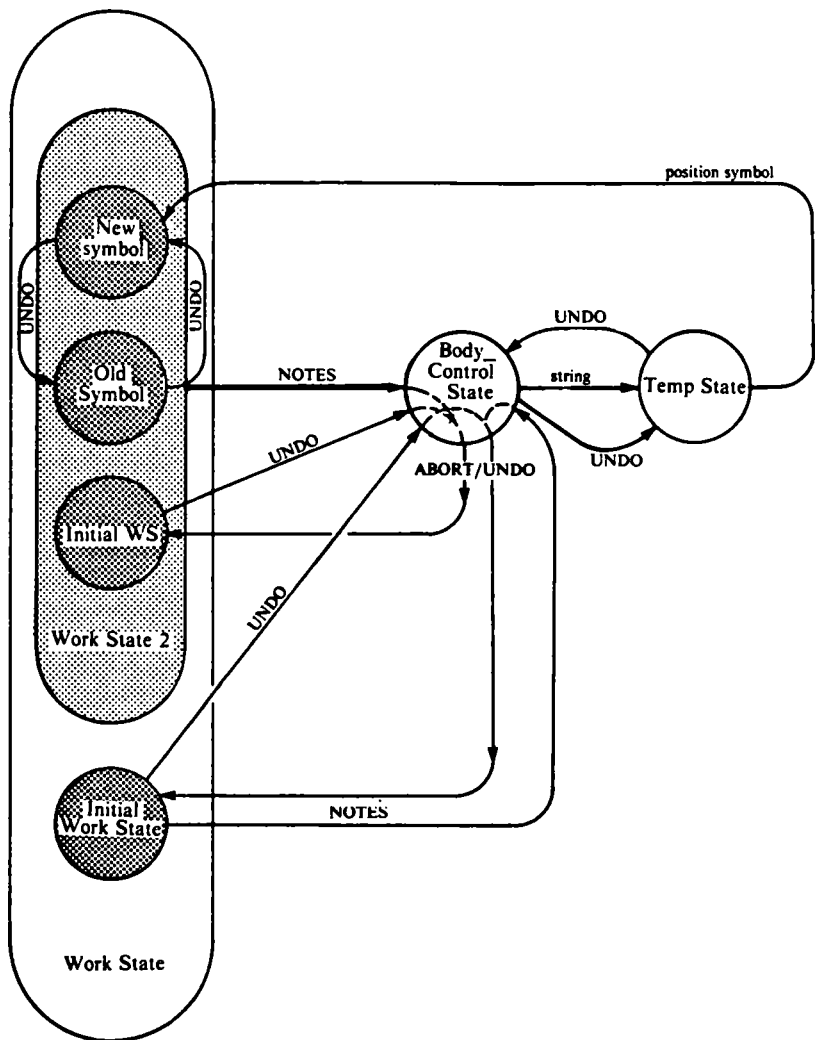


Figure C.21. NOTES command.

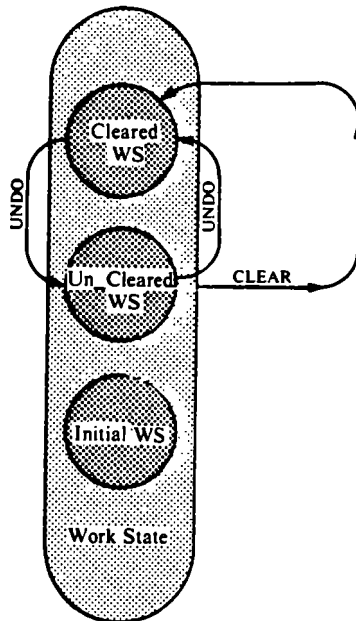


Figure C.22. CLEAR working frame command.

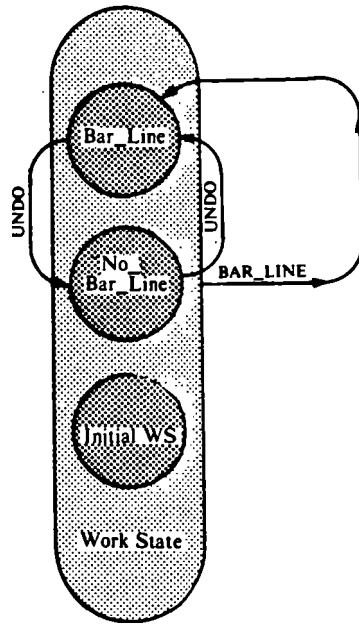


Figure C.23. BAR\_LINE command.

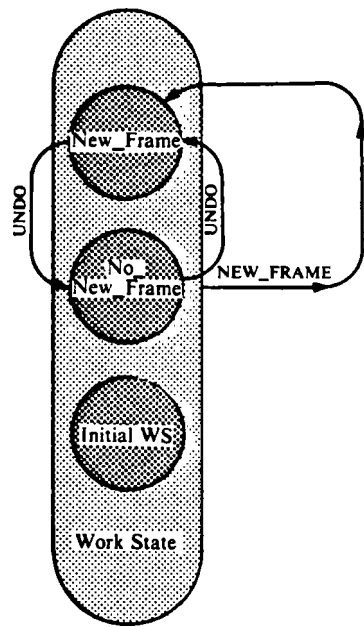


Figure C.24. NEW\_FRAME command.