

ChoreoScribe

A Graphics Editor to Describe Body Position and Movement Using Benesh Movement Notation

by

Detlef O.K. Dransch¹

John C. Beatty²

Rhonda S. Ryman³

This work was originally submitted in fulfillment of the requirements for the degree of Masters of Mathematics to the Faculty of Mathematics of the University of Waterloo by the first author. The research reported here was supported in part by the Natural Sciences and Engineering Research Council of Canada, the Social Sciences and Humanities Research Council of Canada, the University of Waterloo, and the Computer Graphics Laboratory.

¹ Present Address: Bell-Northern Research Ltd., P.O. Box 3511, Station C, Ottawa, Ontario, Canada, K1Y 4H7.

² Present Address: Computer Graphics Laboratory, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1.

³ Present Address: Dance Group, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1.

Acknowledgements

The work described here is by no means a solo effort. It would not have been possible without help, support and encouragement from many people. I would like to take this opportunity to acknowledge a few of them.

Let me begin by thanking my supervisors, Professors John Beatty and Rhonda Ryman. We spent many hours (most of them, patiently) discussing Benesh Movement Notation and the design of the user interface of the system. Thank you for your guidance and input. Also involved in many of the discussions were Robyn Hughes-Ryman and Doug Moen.

Next, many thanks go to John Beatty, Rhonda Ryman, Colin Ware and Jay Black for taking the time to read through rough drafts of this document and suggesting thoughtful improvements.

A great deal of credit goes to the directors of the Computer Graphics Laboratory, the graduate students working in the Lab, the other graduate students and administrative staff in the Computer Science Department at the University of Waterloo for providing a friendly environment to work in and a source of answers to all my questions. Special thanks are extended to Dave Forsey and Dave Martindale for always taking the time to listen to my technical questions and having the right answers.

Finally, there are some persons outside the academic environment who showed a great deal of interest and provided encouragement and support. I would like to express my gratitude to my parents, my sister Loretta and all my friends (especially, Barb, Thomas and Ernie— who also took the time to read a draft of this report and provide corrections and suggestions).

Herzlichsten Dank ... Lebewohl.

Detlef Dransch
University of Waterloo
August 1985

Abstract

Benesh Movement Notation (BMN) is an ideographic language, written on stave lines akin to those used in music, for recording human body positions and movement in three dimensions. Its primary use has been to record a wide repertoire of dances. The preparation and revision of dance scores is a lengthy and error-prone task; this document describes a computerised editor designed to assist in this notation process. The editor employs a medium resolution, colour graphics display as its primary output device and a tablet with puck as the primary input device for a menu-based user interface. The editor serves as a testbed for continuing research in a variety of areas of man-machine interaction and the implementation of effective user interfaces. Topics such as the use of system-initiated and user-initiated dialogues, the design and implementation of a menu-based user interface, and the effective use of feedback are discussed. We describe the current state of a prototype and possible future extensions.

Contents

1 Introduction	1
2 Problem Description	5
2.1 Benesh Movement Notation	7
2.1.1 Musical Scoring Information	11
2.1.2 Position and Movement Information	13
2.1.3 Staging Information	22
2.1.4 Notes	26
2.2 Some Design Considerations	28
3 Editor Description	32
3.1 Current Environment	33
3.2 Screen Layout	37
3.3 Data Structures	39
3.3.1 The Score: Stave Lines and Frames	39
3.3.2 The Frame: Signs and Movement Lines	43
3.4 Some Implementation Details	49
3.4.1 Score Formatting	49
3.4.2 Determining Frame-Types	51
3.4.3 The Percolation of Changes	51
3.4.4 Score Edit Operations and Movement Lines	52
3.4.5 Rendering Movement Lines	53
3.5 Some Interaction Details	56
3.5.1 A Single Symbol Sign	56
3.5.2 A Multiple Symbol Sign	57
3.5.3 Other Methods of Building a Sign	58
3.5.4 A Locomotion Movement Line	60
3.6 Typeset Output	62

4 User Interface Issues	63
4.1 Some Design Issues	66
4.2 A Menu Driven System	77
4.3 Providing Feedback	82
4.4 Editing Curves	87
5 Future Extensions	89
5.1 A Workstation Environment	90
5.2 The Body Model	95
5.3 Extended Notational Functionality	97
5.4 Enhanced Editor Functionality	99
6 Conclusions	103
Appendix A - A Hand-written Benesh Score	105
Appendix B - Implemented Functions	111
Appendix C - Help Menus	112
Appendix D - Versatec Plotter Output	133
Appendix E - Imagen Laser Printer Output	137
Appendix F - Autologic Typesetter Output	141
References	145

List of Illustrations

2.1	The Human Body in Anatomic Position	9
2.2	A Work Frame	10
2.3	Limb Domains in Frame	14
2.4	Various Head Positions	19
2.5	In-Frame Gesture Movement Lines	23
2.6	Direction Signs	24
2.7	Stage Locations	26
3.1	The Hardware Configuration	33
3.2	The Video Generation System	34
3.3	The Screen Layout	38
3.4	Stave Lines and Frames	40
3.5	A Frame Node	41
3.6	Locomotion Movement Lines	46
3.7	Locomotion Line Coordinates	47
3.8	Rendering a Slide Line	54
3.9	Defining a Stage Location	58
3.10	Defining a Turn	59
4.1	A Layered Model for the Code	69
5.1	The Scroll Bar and Marker	99
5.2	Symmetry in Notation	101

1 Introduction

In order to preserve and study accumulated knowledge, we rely on the communication of ideas, concepts and events. In early history this communication was accomplished by the oral transmission of human memories. Clearly, this is a fallible process. Both the acquisition and recall of information are interpretive processes. In addition, what is recalled from memory is often not as complete or accurate as the original information. More recently we have come to rely on recording information in written form for the purposes of preservation and study. Paper and, more recently, microfiche, computer magnetic tapes and disks have become the storage media for much information. Audio tape, motion pictures or video tape are also used for this purpose in areas such as music, film, theatre and current events.

In the written form, concepts and events must be recorded through the use of some type of notation. A common example of a notation is the Latin alphabet which is used as the basis for many languages. Words and sentences are used to express the knowledge to be stored. A second is the form in which music is recorded, namely musical notes, bar lines, rhythm indicators, etc. on a stave (or staff) line. Both these notations are relatively compact, concise mechanisms to express, store and recall specific information. One area in which no concise and economical notation has been in wide-spread use is movement, especially human movement. Recently, much interest has developed in this area for the purpose of recording dance.

Until recently, because no accepted form of notation existed, those aspects of dance which were readily expressible in the written language were primarily studied. This included such things as staging, costuming and the history of a dance. The primary aspect of the dance, human movement, was not recorded. Obviously, expressing movement in a written language would be quite

impractical, cumbersome and involved. In fact, some dance notation systems exist which are based on written descriptions, for example the system developed by Saunders [Saunders46]. However, many other systems developed in the twentieth century are based on graphical signs and symbols. Some of the better known of such systems include Benesh Movement Notation [Benesh77], Labanotation [Laban75], and the Eshkol-Wachmann system [Eshkol58]. Guest provides us with an insightful analogy between dance notation and other forms of notation: "Dance notation is (or should be) to dance what music notation is to music and the written word to drama" [Guest84]. We shall often return to these analogies throughout our discussion. A good discussion of the use of notation versus video tape or film may be found in [Benesh77].

Just as computer systems have been applied to the entry, manipulation and storage of text using editors, text formatters and wordprocessors, so this report describes an analogous application of computer science to the area of dance notation. The "alphabet" is that of Benesh Movement Notation developed by Rudolf and Joan Benesh. The notation is the alphabet and each practical application of it to such areas as classical ballet, modern jazz and folk dance forms the corresponding language of discourse [Benesh78]. *ChoreoScribe* is a computer system being developed in the Computer Graphics Laboratory at the University of Waterloo in cooperation with the University of Waterloo Dance Group, the National Ballet of Canada and the Institute of Choreology in London, England. The design of *ChoreoScribe* (from the Greek word *khoreia* for dance and the Latin word *scriba* for write) has been directed to some extent by the language of classical ballet in that some aspects and functions of the system deal with ballet idiosyncrasies.

This project is a continuation of work done by Baldev Singh [Singh82, Singh82a, Singh83]. The objective of Singh's work was to determine the feasibility of providing an editor for the creation, manipulation and archiving of a ballet score using Benesh Movement Notation. The work described in this report is a direct extension of Singh's prototype and is another step toward a full production system. Many interaction techniques of the original editor have been retained and applied to new functions implemented in the current version. Where new types of interaction were required for new functions, they were designed to be consistent with those developed in the earlier prototype. A major objective of this version was to provide enough facilities to allow the notation of a ballet solo. To this end, modifications were made to functions of the earlier

prototype and new features were included in the current implementation of the editor. These include:

- formatting the score— this required an extension to the data structures used to store information and the implementation of a proportional spacing algorithm for frames on a line;
- substantially extending the Benesh vocabulary available in the editor— in addition to making more Benesh signs available (from 375 in the previous version to over 2200 in the current version), this also included the implementation of movement lines (using B-splines);
- implementing four frame types to facilitate the development of an appropriate proportional spacing algorithm, and to accommodate the notation aspects and the body model aspects of the editor (e.g. the percolation of data).

In addition to the editor itself, a separate program has been developed to facilitate the production of high quality hardcopy output suitable for publishing Benesh scores created using the editor. This work has been carried out by Renata Kraszewski and John Chapman.

ChoreoScribe has been developed and runs in a research environment utilising high-performance, high-cost hardware. However, the system only uses a subset of the hardware environment which is likely to be widely available in an inexpensive workstation in the near future. It is expected, as hardware costs decrease, that it will be feasible to move a production version of the editor to a low-cost workstation environment. It is, of course, imperative that such an environment be found for the editor before it can find wide acceptance in the ballet community.

As a first step in this direction, a parallel effort is also underway at the University of Waterloo to implement a Benesh editor on the Apple Macintosh personal computer. Work on this program is being done by Doug Moen. Partially influenced by the environment, a different approach has been taken to the design of the user interface of the Macintosh editor. It has been influenced, in part, by general design principles employed by a great deal of other software developed for that environment. This has in turn influenced, to some extent, the development of the user interface of the system described in this report.

As well as being a useful system for the historical preservation of dance through notation, as a tool for the study of dance choreography and as a tool for teaching Benesh Movement Notation, ChoreoScribe is a vehicle with which to explore some areas of computer science such as the design of human interfaces in computer systems. As the system develops further, it should also provide insights into such areas as human figure animation and robotics.

In Chapter 2 of this report a brief synopsis of Benesh Movement Notation and a brief description of some design issues and considerations are provided. Chapter 3 contains a description of the hardware environment in which the editor currently runs. This is followed by a description of the current version of the editor, which includes a look at the data structures, some implementation details and some interaction techniques used. In Chapter 4 we explore some of the human interface and software engineering issues encountered during the development of the editor. Chapter 5 provides some suggestions for further extension of the editor.

2 Problem Description

A particular ballet is an evolving, changing entity. The history of such a work often consists of numerous productions. In many of these productions additions, deletions and alterations are made by choreographers, musicians, directors and performers. The reader is referred to [Cohen82] for a description of the history of Swan Lake, for instance, from its first production in Moscow in 1877, through five years of changes as part of the repertory of the Bolshoi Theatre, to its reworking in 1894 for a production in St. Petersburg in 1895. But how has the choreography changed from one production to another? Unless lengthy, verbose, written descriptions exist, such detailed information is lost. The invention of various dance notations has provided more efficient tools for the historic preservation of such detailed choreographic information. One such system is Benesh Movement Notation.

Since its invention, Benesh Movement Notation has been adopted by many dance companies throughout the world, including the National Ballet of Canada. Many have added a *choreologist* (Benesh notator) to their staff. One of the functions of the choreologist is to record the dances produced by the company for historical preservation. They are also often involved in the reconstruction of a dance from a written score. Let us look at the typical life-cycle of a dance score.

The process begins with the choreologist attending rehearsals and making rough notes as the choreographer works with the dancers. These rough notes are gradually refined and additional notes made during subsequent rehearsals. During these rehearsals a choreographer may decide to add or drop parts of the dance or revise certain sections, perhaps to better suit a particular dancer's style or strengths, or perhaps for other artistic reasons. It is from this collection of rough notes that the choreologist creates the final version of the score. The resulting score is checked for completeness and correctness. It is then given to

an *autographer* who rewrites the score with special drafting pens. The score is now ready to be archived, for later reproduction.

When a dance is to be reproduced from an existing score, the choreologist first works with the dancers. Subsequently the choreographer or repetiteur may refine certain movements according to an individual dancer's style. Modifications may also be made by the choreographer for this particular production of the dance. These modifications are noted by the choreologist and eventually a revised score containing the modifications made is produced.

Clearly, the amount of information the choreologist must deal with is enormous. The actual process of creating a Benesh score is a tedious task involving many hundreds of hours. An efficient mechanism for creating and verifying, storing and retrieving, modifying and re-archiving a finished score would be an invaluable tool. It is not uncommon to have changes made to a performance once a Benesh score has been completed. This may result in modifications not only to the bars involved, but to many pages of the score due to the spacing of information. The production of publishable scores directly from the stored information would also be of great benefit.

We begin our discussion with a brief overview of the Benesh Movement Notation and then turn our attention to a computer tool designed to assist in the creation of a Benesh score.

2.1 Benesh Movement Notation

This notation system, devised by Joan and Rudolf Benesh in 1947 and made public in 1955, is one of the leading systems used for scoring classical ballet. In its design Rudolf Benesh took an artist's point of view, believing that because the perception of movement is visual, the form of notation representing it should also provide a *visual image*. As a result, Benesh Movement Notation is based on a graphical matrix. The five line stave, used in the sister art form music, provided an ergonomically ideal matrix in which to represent the human figure. The design of signs to specify various types of information such as travel, direction faced and turns also reflects the principle of providing a visual representation. Similarly, the implementation of movement lines is consistent with this principle.

Another principle of Benesh Movement Notation is that of *redundancy avoidance*. If one piece of information can be deduced from the presence (or absence) of one or more others, that piece of information is not written. An important concept of the Benesh notation is that only body parts that have changed position from one body posture to the next are notated. The same is true for staging information such as direction faced and stage location. If something is not specified at a particular point in the score, the most recent definition of the item in question is assumed.

Finally, the system was designed to be conceptually simple, yet very flexible. It is from these two attributes that Benesh Movement Notation derives its power to express clearly the many forms and styles of human movement.

In the following discussion, let us define the *anatomic reference system* used in body description. In the anatomic position the body is erect, facing forward with arms at the sides (see Figure 2.1). We shall discuss body positions with respect to the following planes [Logan77]:

Sagittal - any plane dividing the body vertically into right and left portions; the *Cardinal Sagittal Plane* divides the body through the mid-line into left and right halves.

Coronal - any vertical plane at right angles to the sagittal plane that divides the body into anterior (towards the front) and posterior (towards the back) portions; the *Cardinal Coronal Plane* divides the body into front and back halves.

Transverse - any plane dividing the body into superior (above) and inferior (below) portions; the *Cardinal Transverse Plane* divides the body into top and bottom halves.

The notation is written from left to right on a five line *stave*, just as for music. Each stave describes a number of static body positions, as if one were to take a series of 'snapshots' of the performer. Each such static body position corresponds to a beat (or a sub-beat) of the accompanying music. Each snapshot is known as a *frame* (see Figure 2.2). A frame uses the five lines of the stave and space above and below to present additional information. The five lines of the stave conveniently correspond to the top of the head, shoulders, waist, knees and the floor when the body is in the anatomical position. These points of intersection provide a constant frame of reference when specifying various body parts. With the arms extended sideways at the shoulder, the arm span approximates the height of the body and thus the basic frame width approximates the distance from line 1 to line 5 in Figure 2.2. In addition to the five lines of the stave, the editor described here shows a *leger line* appearing above the stave lines to delimit the maximum reach of the hands when the arms are fully extended upward. A corresponding leger line is provided below the stave lines as a guide for the placement of signs. The *centre line* of the frame is also shown. It indicates a division of the body into left and right halves, regardless of body posture. It is not the line of gravity, although these two do coincide when the body is in its anatomic position (Figure 2.1).

The body position and movement must be recorded as they are seen. A viewpoint must therefore be established. Two obvious choices are (i) from directly in front or (ii) from directly behind the performer. One is the reflection, or mirror image, of the other. Note that both of these are relative to the performer and not relative to the stage (e.g. backstage or audience). For the purposes of the notation, the viewpoint is chosen to be directly behind the performer. This has two practical advantages. The first is that when notating a performer's right arm, it appears to the right of the centre line in the frame. Secondly, the reader of a score readily identifies with the performer; the reader's right is the performer's right also [Benesh77].

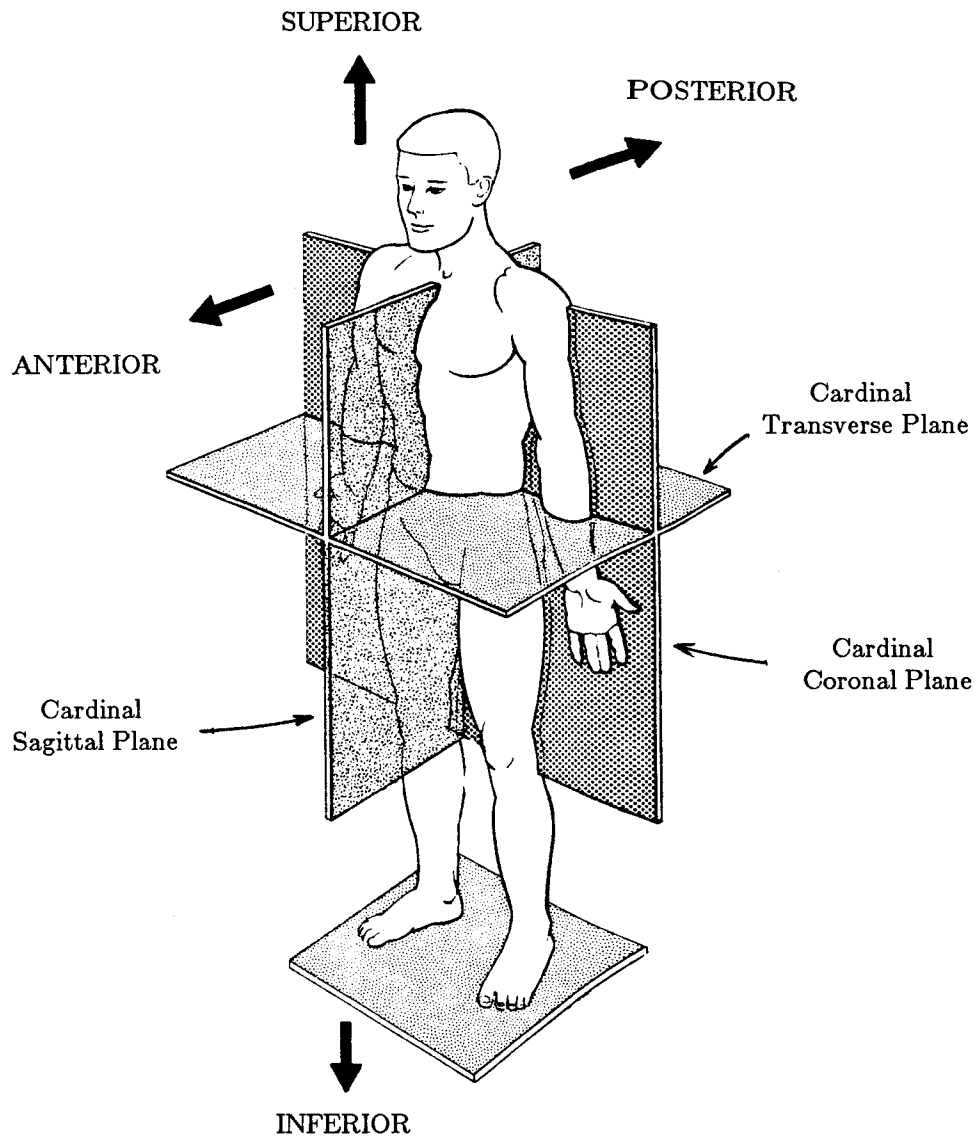


Figure 2.1 The Human Body in Anatomic Position

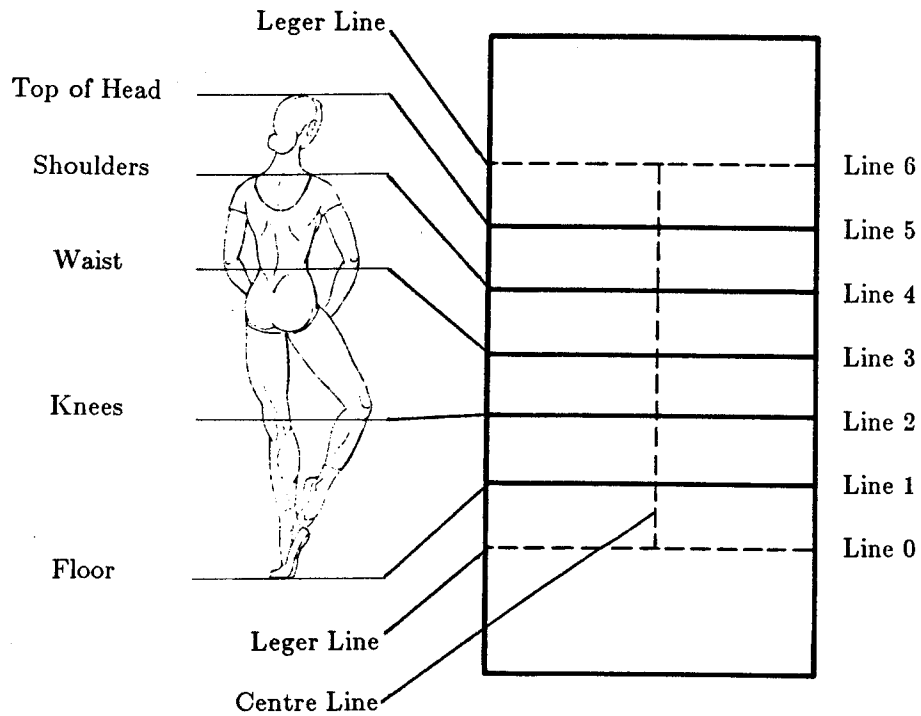


Figure 2.2 A Work Frame

Basic body positions are recorded by noting the projection of the four extremities (hands and feet) and the bends (elbows and knees) onto the cardinal coronal plane. What is notated however, is not the projection of the body as a whole. Rather, it is the projection of the body segments in relation to the body mid-line into the corresponding areas between the stave lines. For instance, if the arms are fully extended sideways at the shoulders and the pelvis and torso are tilted forward such that the shoulders are seen at waist height, the hands are not notated at line 3 in the frame but rather at line 4.

Although the editor has been designed to allow notation of body position and movement in any context, specific details are incorporated in order to allow its application to ballet. As such, additional facilities to allow specification of musical scoring and staging details have also been included. Information described using Benesh notation may be categorised as follows:







- (i) musical scoring information,
- (ii) body position and movement information,
- (iii) staging information, and
- (iv) notes.

2.1.1 Musical Scoring Information

As in music, a numeral is placed in the top two spaces (between line 3 and line 5) of the staff to indicate the number of beats in each bar of the score that follows. This is known as the *time signature*.







A single *bar line* marks the beginning and end of each musical bar. If a bar must be split over two staff lines of the score, that is, if some frames of the bar appear in one staff line and the remaining frames of the bar appear on the subsequent staff line, a dashed bar line is placed at the end of the first staff line and the beginning of the second. Otherwise, a single bar line always appears at the beginning of a staff line. A double bar line denotes the end of a sequence.

Repeat signs are used to indicate that a series of frames is to be repeated. As in music, a repeat sign consists of a double bar line in which the outer line is slightly longer than the inner one. Unlike music, a series of repeated frames may be qualified in numerous ways. A sequence of postures and motions may be repeated in exactly the same manner, may be repeated on the other side (i.e. reflected in the cardinal sagittal plane), or may be repeated front to back (i.e. reflected in the cardinal coronal plane).

	identical repeat (same side)
	symmetrical in left/right repeat (other side)
	alternating repeat (first other side, then same side)
	symmetrical front/back repeat (reverse; other direction)
	dotted repeat (identification of a repeat that includes similar repeats)
	double dotted repeat (identification of a repeat that includes dotted repeats)

The number of repeats is indicated by a numeral immediately following the repeat sign in the centre two spaces (between line 2 and line 4) of the staff. The digit '3', for example, indicates that the noted bar(s) is repeated three times (i.e. it is performed four times in all).

A particular position described in a frame need not always be reached on a full beat in the bar. A beat may be divided into parts and these indicated with *rhythm signs*. The various signs illustrated are used to specify precisely when a body position is reached relative to a beat of the music.

	<i>pulse beat</i>	specifies a full beat or count
	<i>te</i>	specifies 1/4 beat after the count
	<i>an</i>	specifies a beat half way between counts
	<i>ti</i>	specifies 3/4 beat after the count
	<i>dai</i>	specifies 1/3 beat after the count
	<i>dee</i>	specifies 2/3 beat after the count

2.1.2 Body Position and Movement Information

We have seen that notating body positions is based on the projection of various body parts onto the coronal plane. This provides the reader with two-dimensional information. To completely define the position of a limb, we must know if it is *in front of*, *level with* or *behind* the body wall. The *body wall* may be visualised as an area surrounding the coronal plane equal in depth to the thickness of the body. To provide the information required to define the third dimension (the sagittal, forward-backward direction), different signs are used for the limb in question. The following *basic signs* denote an *extremity* when it is:

	in front of the body wall
—	level with (within) the body wall
●	behind the body wall.

The position of the sign in the frame uniquely defines the exact location of the extremity. It is not necessary to further describe how far in front of or how far behind the body wall the extremity is located because only one position is physically possible, assuming limbs are of fixed length and attached to the body at specific locations.

Similarly, the *basic signs* to denote a *bent elbow* or *bent knee* are:

+	in front of the body wall
+	level with (within) the body wall
X	behind the body wall.

Again, the amount of bend is defined by the position of the sign in relation to the extremity sign and human anatomy.

The reader will note that the basic limb signs are identical for left and right, for hands and feet, and for elbows and knees. The specific limb being defined is usually inferred from the position of the sign in the frame. This is, however, not sufficient. Consider an example in which the left hand crosses the body mid-line

and is positioned next to the right hand. Both signs would be identical and one would not be able to discern which defines the left hand and which the right. As a result, each limb has a corresponding *domain* in the frame as illustrated in Figure 2.3.

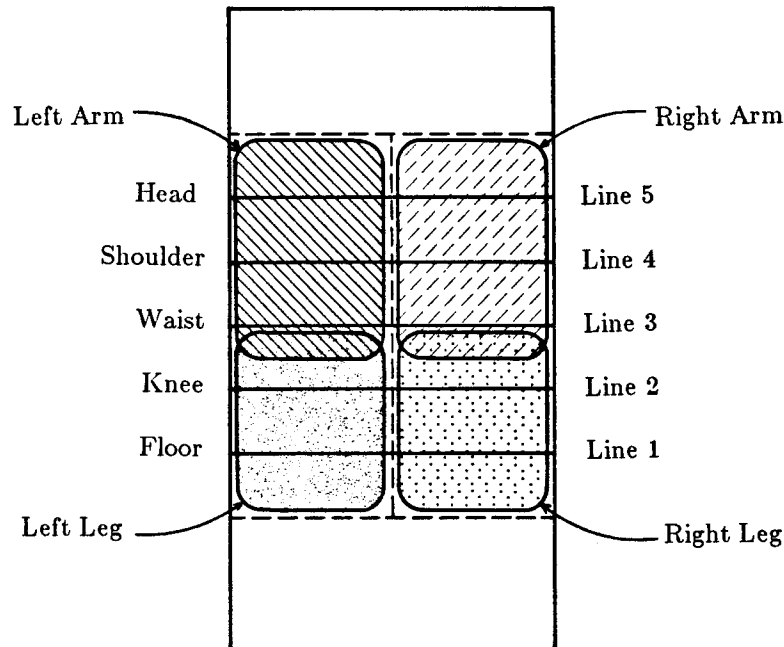
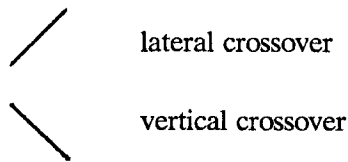


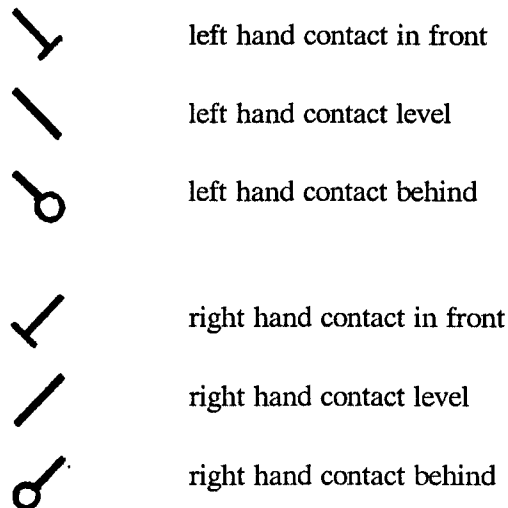
Figure 2.3 Limb Domains in Frame







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: the *lateral crossover* is written over a sign to indicate that an extremity or bend has moved over to the opposite side of the body (left-right) and a *vertical crossover* to indicate a hand or elbow below the natural hand position (half way between line 2 and line 3) or a foot or knee above line 3. The crossover signs are:







Crossovers are not necessary when the context clearly specifies the body part. Such is the case, for example, when the feet are below the hands or when a movement line clearly indicates which extremity/bend has crossed over. When crossovers are written, they are shown using a finer line than that used for other signs in order to distinguish them from, say, a bent elbow behind the body wall.





The notation also defines signs to indicate that an extremity *contacts* another part of the body. The sign itself indicates which limb is being specified and the placement of that sign in the frame specifies which part of the body it contacts. For example, if the right hand contacts the left shoulder at the front of the body, the appropriate sign (right arm contact in front) is placed at line 4 of the stave to the left of the centre of the frame. The various contact signs are:











	left foot contact in front
	left foot contact level
	left foot contact behind
	right foot contact in front
	right foot contact level
	right foot contact behind





In addition to the basic signs used to define individual feet, Benesh Movement Notation provides a number of signs to specify certain positions of *both feet*. In classical ballet these are (i) *first position* in which the feet are in a straight line, heels touching, (ii) *third position* in which one foot is in front of the other, the heel against the instep, and (iii) *fifth position* in which one foot is in front of the other, the heel against the joint of the big toe [Clarke77]. The *closing* foot is the one which is moved to attain the indicated position. The signs for these positions are:

	first position, static
	first position, left foot closing
	first position, right foot closing
	first position, both feet closing

-  third position, left foot behind, static
-  third position, left foot behind, left foot closing
-  third position, left foot behind, right foot closing
-  third position, left foot behind, both feet closing




-  third position, right foot behind, static
-  third position, right foot behind, left foot closing
-  third position, right foot behind, right foot closing
-  third position, right foot behind, both feet closing

-  fifth position, left foot behind, static
-  fifth position, left foot behind, left foot closing
-  fifth position, left foot behind, right foot closing
-  fifth position, left foot behind, both feet closing

	fifth position, right foot behind, static
	fifth position, right foot behind, left foot closing
	fifth position, right foot behind, right foot closing
	fifth position, right foot behind, both feet closing

Each of these signs may be tilted to show that the heel on the raised side is lifted. In addition to the “flat” position, two discernible degrees of tilt in both directions are generally sufficient. Finally, the position of the sign relative to the floor line (line 1) also imparts information to the reader.

Head, torso and *pelvis* positions are all defined by the use of signs consisting of one or two straight line segments. The signs used for these three *main body parts* and their meanings are identical. However, the head sign always appears in the uppermost space of the stave (between line 4 and line 5), the torso sign between line 3 and line 4, and the pelvis sign between line 2 and line 3. Each of these body parts is given three types of displacement in the notation: *tilt* to the left side or the right side, *turn* (or rotation) to the left or the right and *bend* backwards or forwards. The tilt is described by the angle of the long stroke of the sign. The turn is specified by a shorter stroke perpendicular to the tilt-stroke. The length of this stroke indicates the degree of turn. Finally, the intersection point of the tilt-stroke and the turn-stroke defines the extent of backward or forward bend. If no turn and no bend is to be specified, the short stroke is omitted from the sign (only the tilt-stroke is shown). Each type of displacement has seven basic positions in the notation as illustrated.

tilt:	left		right
turn:	left		right
bend:	backward		forward

A main body sign may be composed of any combination of the above choices. Examples of a few head signs are presented in Figure 2.4 [Singh82a].

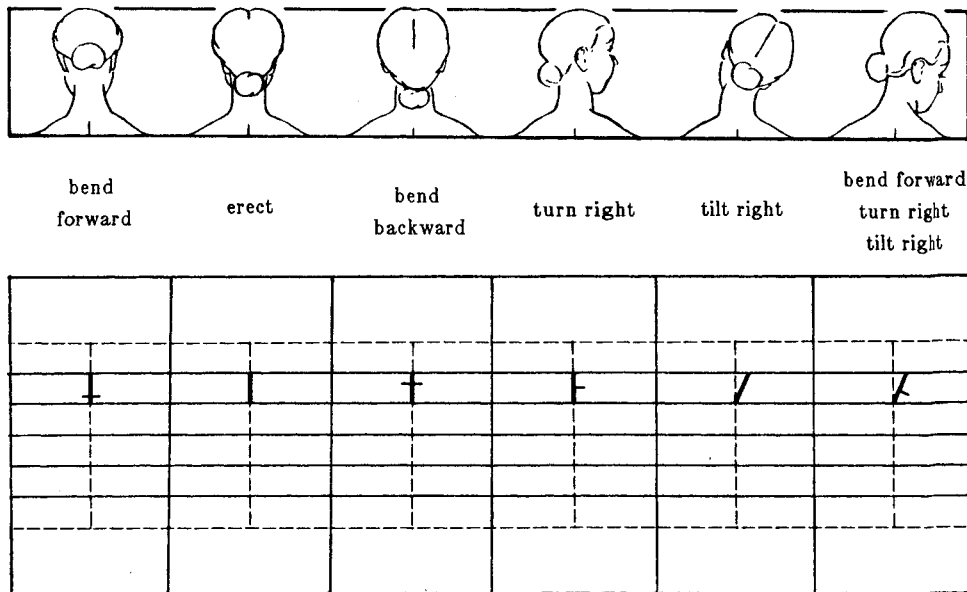


Figure 2.4 Various Head Positions

To this point in our discussion we have examined the means used to describe static postures. The change in posture from one frame to the next indicates that some movement must take place. However, usually the movement to be described is not a simple repositioning of a body part but may, for instance, include a circling motion. Benesh notation makes use of *movement lines* to record specific details about the transition between static postures. Movements may be gestures or locomotion and may be drawn within a single frame or between frames.

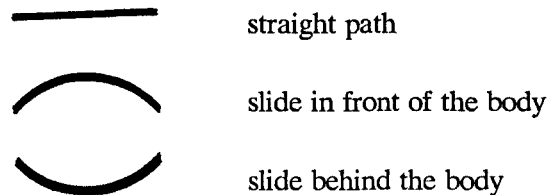
Between-Frame Locomotion Lines

Benesh Movement Notation uses between-frame movement lines to represent most forms of *locomotion*. In effect, one static position is linked with another and the link describes the type of locomotion that takes place between the two. Benesh Movement Notation describes three kinds of locomotion: steps, jumps and slides. Such movements can be made in any direction but, regardless of the direction travelled, the movement line is written from left to right in the stave line (the chronological order of the postures described in the frames).

To indicate a step, a curved *step line* is drawn from the frame in which the step begins to the frame containing the final posture. The line is drawn to indicate the path of the foot during the step. If the step is a high one for instance, the movement line is drawn from the vicinity of the initial position of the foot, with an apex between the waist line (line 3) and the knee line (line 2) of the stave, and its end is attached to a sign indicating the foot with which the step was taken. To indicate a low step, the step line is drawn close to line 1 of the stave. In any case, the step line always appears above the floor line.

A *jump* is traced as a movement line drawn under line 1 of the stave. The path traced can be thought of as the reflection of the jump, as if the performer were dancing on a mirror. Hops and bounces are special cases of a jump and are discussed later.

A *slide line* is also written below line 1 of the stave. However, to accommodate the curved slides which occur in classical ballet (such as *ronds de jambe*), these lines may consist of straight and/or curved parts. Sliding in a straight path is indicated by a horizontal straight movement line. To show slides in a transverse plane, the shape of the slide line is altered as follows. To indicate curved slides in front of the body an upward curve is used and to indicate curved slides behind the body a downward curve is used.



A single slide may consist of any combination of these possibilities. That is, it may be composed of a number of these shapes placed end to end.

To complete the specification of locomotion lines one must be able to specify the direction of travel. To specify a forward movement, the movement line is attached to (for steps and jumps) or ends under (for slides) the centre of the foot/feet sign and is qualified with an in front sign (a short vertical stroke). Conversely, if the movement is backwards, a behind sign (a dot) is used to qualify the line.

Travelling sideways is indicated by the attachment of the movement line relative to the foot/feet sign. To indicate locomotion to the left, the movement line is attached to or ends under the right side to the foot/feet sign. The fact that the foot/feet sign appears to the left of the end of the movement line indicates travel to the left. Conversely, if the foot/feet sign appears to the right of the end of the movement line, travel is to the right.

Finally, diagonal travel adheres to the latter rule for attachment and a modified qualification is used to specify the forward or backward component of the movement. The qualifications used here are appropriately placed either to the right or the left of the centre of the movement line.

In-Frame Locomotion Lines

Steps, jumps and slides may also be notated using in-frame movement lines called *contracted locomotion lines*. These are locomotion lines as described above except that the starting position of such a line is at the left side of the same frame in which the line ends.

Hops and *bounces* are considered to be special cases of jumps that commence just prior to and finish on a beat (or sub-beat). These are indicated with a special type of contracted jump line that appears as a small semi-circular sign written under the appropriate foot/feet sign. These signs may also be qualified with a stroke or dot as described above.

In-Frame Gesture Lines

An *in-frame gesture line* traces the path of a body part from the starting position to the final position. The line begins at the position in the frame where that body part was last notated, and ends attached to the sign specifying its final position. This defines the direction of the movement. If the apex of the path is forward or backward of the body part's starting and ending point, a *qualification* is placed on the movement line. A short stroke drawn perpendicular to and centrally placed on the movement line indicates a forward excursion while a similarly placed dot describes a backward excursion. Movements in a coronal plane do not require any qualification. An example of some in-frame gesture movement lines are shown in Figure 2.5.

Between-Frame Gesture Lines

Between-frame gesture lines may also be used to trace the paths of gesturing extremities when these cannot be clearly shown within a single frame.

2.1.3 Staging Information

We now turn our attention to defining the staging information required to notate dance. Signs to specify directions faced, turns, travel and stage locations are written below the stave lines.

One of eight basic signs may be used to indicate which *direction* the performer is facing. The dot of the sign may be thought of as the head of an arrow which points in the direction faced. The direction signs are illustrated in Figure 2.6.

To specify that a performer *turns* we require a sign which indicates:

- the direction faced at the start of the turn,
- the direction of the turn (clockwise or counter-clockwise),
- the direction faced at the end of the turn, and
- the amount of the turn (e.g. $\frac{1}{2}$ turn, $1\frac{1}{2}$ turns, etc.).

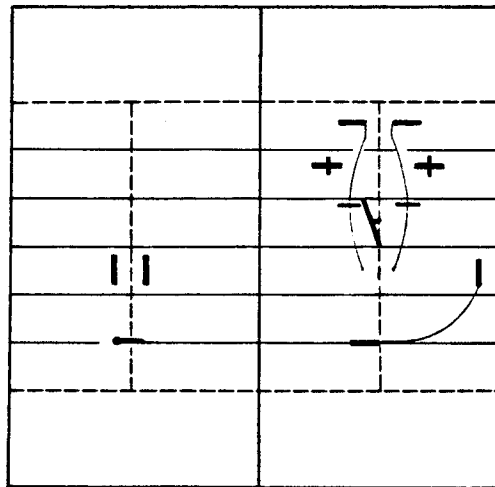
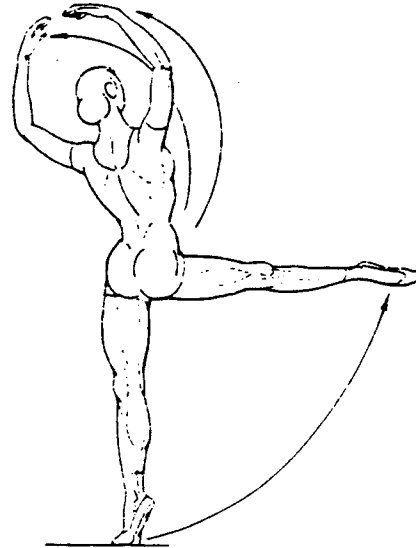


Figure 2.5 In-Frame Gesture Movement Lines

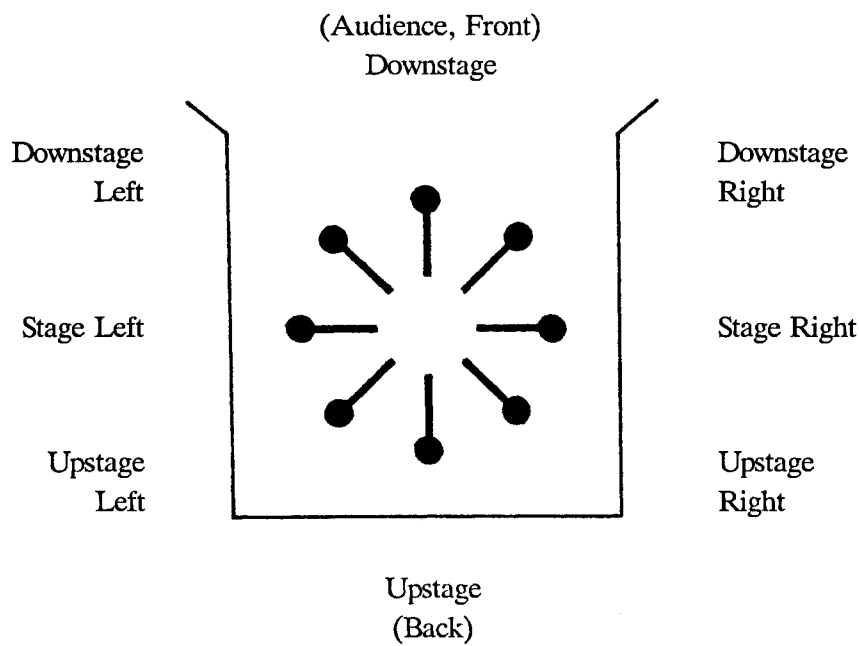
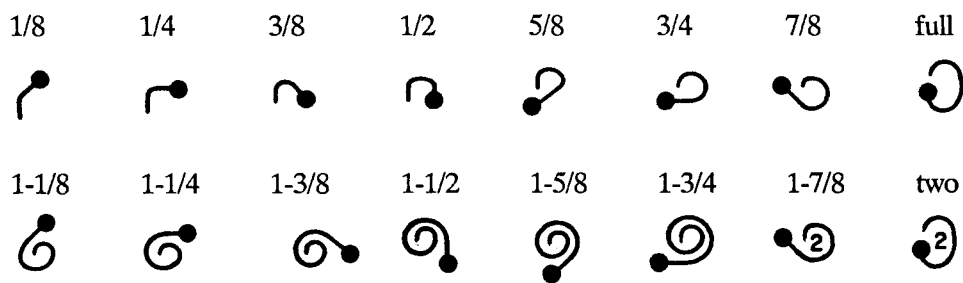


Figure 2.6 Direction Signs










An extension of the direction faced sign is used to serve this purpose. The sign is begun in the starting direction faced, and a spiral line is sketched clockwise or counter-clockwise that ends in the final direction faced with a dot (like a direction sign). Some turn signs are illustrated below:



The turns shown here all begin facing downstage. There are seven other

possibilities for the beginning direction (see directions faced). For these the appropriate sign is rotated accordingly. Also note that we have only shown clockwise turns. The corresponding counter-clockwise turns are notated with the spiral turning in the opposite direction. To indicate a turn movement involving more than two circles, the numbers '2', '3' or 'x' (for an unspecified number) may be placed in a turn sign of the second row above.

Travel signs are more complicated than most static signs since two directions must be incorporated into one sign: the direction faced relative to the line of travel and the direction travelled relative to the stage. The general sign used to indicate travel is an arrow. The various possible directions faced (relative to the line of travel) are indicated by different tick marks of the arrow. The various signs are:

	general travel
	forward
	backward
	sideways left
	sideways right
	diagonally forward left
	diagonally forward right
	diagonally backward left
	diagonally backward right

When the direction faced has been determined, the arrow is pointed in one of the eight basic orientations discussed under the direction faced signs.

In order to specify the whereabouts of the performer on the stage a *location sign* is used. Each one of nine basic stage location signs corresponds to a column of the performing area as illustrated in Figure 2.7. A small horizontal stroke is

placed on the required basic sign to indicate the depth of the position on the stage at which the performer is located. Placing the stroke near the top of the sign indicates a location downstage and one near the bottom indicates a location upstage. When the performer is slightly to the left or right of the centre of the column specified, the stroke is joined to the left or right of the sign accordingly. If the dancer is between two adjacent stage lines, the two adjacent basic signs are written and the stroke placed between them as appropriate.

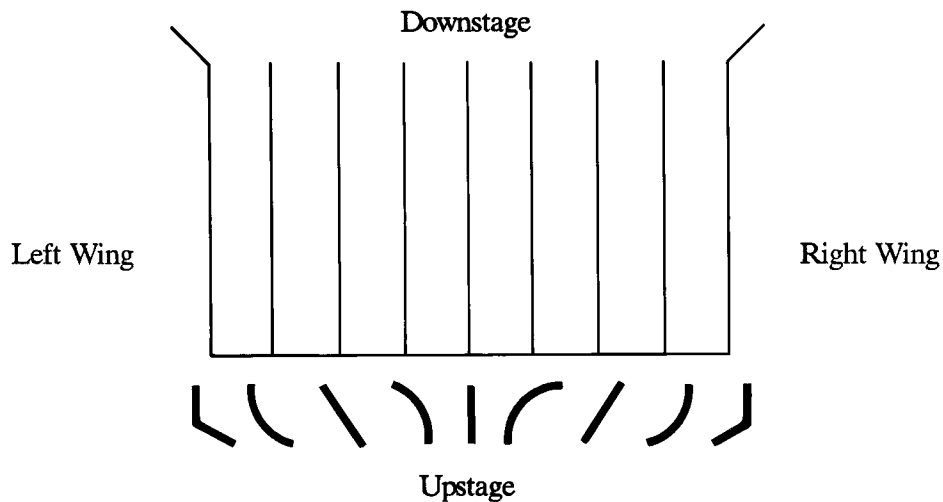


Figure 2.7 Stage Locations

2.1.4 Notes

There are some situations in which text (letters or words) are necessary in a Benesh score. Such *notes* may be used, for instance, to title a score or sequence of frames, to specify the degree of effort in movement (e.g. *pp* - very softly, *f* - strong, etc.), to specify the prevalent quality of movement (e.g. *Andante* - medium slow, *Vivace* - lively, *Dolce* - sweetly, *Pesante* - heavily, etc.), or to indicate something not expressed in the notation (e.g. "thinking of her mother-in-law").

The preceding discussion has been a very brief synopsis of those aspects of Benesh Movement Notation accommodated by the editor in the current version. Some facets of the notation, such as specification of eyes, hand orientation, fingers, etc. have not been discussed here. For a more detailed description of the

notation or the underlying theory the reader is referred to [Benesh56, Benesh77, Ryman82, McGuiness-Scott83, Brown84, Ryman84].

Appendix A contains a portion of a hand prepared Benesh score and a brief description of the information contained in the first few bars.

2.2 Some Design Considerations

Before examining the implementation of the editor in greater detail, let us review some of the major concepts and features considered during the design phase.

One major concept vital to a thorough understanding of the editor is a *duality* in the overall design. In the first place, the system is used to create, modify and archive Benesh scores as they are notated manually today. This involves the creation and placement of signs in the score and the generation and manipulation of movement lines. In function this is analogous to the use of a text editor to create a letter or a report and is the aspect emphasised in the Macintosh project. It deals with the syntax of the notation. Secondly, however, the system described here supports the concept of a full body model. If, for instance, three 'in front' signs appear in a frame, the editor knows which one defines the left arm, which one the right arm and which one specifies a certain foot. Each sign placed in the score is associated with an entity known to the editor (a specific body part, stage location, rhythm, etc.) and a notated movement (in-frame or locomotion) is associated with the body part undergoing the transition. Thus, the editor, to some extent, understands the semantics of the ballet language being implemented. The Benesh editor is analogous to a text editor able to point out errors in spelling, grammar and sentence structure.

As we have seen previously, one of the principles of the notation system is the avoidance of redundancy. If an arm has not changed position since the performer's last posture, it is not notated. If a body part is not defined in a given frame, a choreologist scans backward through the score to find the most recent definition of the item in question to determine its current position. To use this approach in the editor would have resulted in unacceptable computational overhead, often resulting in poor response time. Instead, the approach taken is to store all pertinent information in each frame of the score. If the rules of the notation system specify that a sign not appear in a certain frame (because it is identical in type and placement to a sign in the preceding frame of the score) it is not displayed, even though it is stored in the frame. In other words, at any point in the score the editor maintains a full description of the body model and staging information but displays only that information allowed by the rules of the

notation system. This duality has profound affects on the implementation of the editor and has given rise to some other design principles.

We have just described two distinct modes in the operation of the editor. Normally the choreologist's primary objective is to use the editor to create a score according the rules of the Benesh system and thus for the most part is only interested in the *notation mode* of the editor. In some cases however, the user may not understand how this mode interacts with the *full body model mode*, or may simply wish to see the composite body position at a particular point in the score. Rather than forcing the reader to scan backward to accumulate that information from the data displayed, a switch (menu option) has been provided to show the cumulative body position of the current frame. Effectively, this menu option switches back and forth between these two modes.

One of the basic philosophies of the editor is the prevention of errors in the notation wherever possible. Since the editor has a limited understanding of the syntax and semantics of Benesh notation for ballet, it was decided to use this knowledge to allow only those options that are syntactically and semantically valid in a particular context. Examples of this include: displaying only those menu options that are valid at a given point in time, allowing only valid placement of the various Benesh signs, and permitting only the valid definition of steps, jumps and slides. This is an effective method of error avoidance.

As part of this implementation of the editor, it was decided to create an external database of the many symbols used to represent or construct Benesh signs. As the editor is initiated, this database is read and an internal graphical representation of these signs is produced. It is hoped that this may, in the future, provide a more direct link between the editor and any system used to produce hardcopy output of a score since they could both create Benesh signs from a single database. This scheme will also facilitate the interactive editing of symbols in the database.

In addition to the creation of the above graphical representation of symbols, an internal representation is also created. This contains the instructions used to produce the graphical representation. The purpose of this internal representation is to facilitate symbol manipulation (such as rotation and menu creation) as these are required while the editor is running. For instance, it is infeasible to store graphically all possible orientations of all the signs and all possible menus of signs. As a result, when a certain variation of a sign (for example, by rotation),

or menu is required, the internal representation is used to create it.

Although we have been using the term “editor” to refer to the system being described in this report, we should spend some time defining what this term means. The term “editor” is normally used to refer to programs which facilitate entry, modification, storage and retrieval of text [Gardner80, Joy83]. The editor itself often knows nothing about the format of what is being entered. The paragraph of a report and the subroutine of a program, although very different in format, may be entered using the same editor. It does not enforce a specific format on the output it produces. Further, text formatting programs exist which take, as input, data that has been created and stored using an editor. This data usually contains embedded commands describing the format the output should eventually take [Ossanna76, Lesk78, Knuth83]. The text formatter creates output according to these instructions. In recent years a number of “editors” have emerged which combine these two functions in one program [Johnson84]. As well as providing the editing functions described above, these editors also do formatting ‘on-the-fly’. This is possible when the editor knows what type of output it is to produce such as, in the case of word processors, sentences, paragraphs, chapters, letters, documents, etc. These are sometimes called “what-you-see-is-what-you-get” (or WYSIWYG) editors. This is the case with the Benesh Movement Notation editor described here, which attempts to format the Benesh score as it is being created and modified. This process involves the placement of bars on lines of the score and the spacing of frames on a line. The editor makes a reasonable attempt to format the data according to a given set of rules described in the next chapter. If the user is not satisfied, the editing operations of the system, such as Copy, Move and Delete frames, may be used to re-arrange the score.

Early in the design phase it became apparent that, due to the flexibility of the Benesh system, it would be a large task indeed to incorporate all possible Benesh signs in the editor. The concept of a *sketchpad* to allow the user to build any conceivable sign from an available library of symbols emerged. This idea has not been implemented in this version of the editor, though it is discussed later. It was decided instead to provide as many signs as possible in their proper context. This was done to remain within the confines of the semantic understanding (full body model, etc.) of the editor. It has become increasingly clear that the sketchpad idea will be a necessary part of any Benesh editor due to the dynamic and evolving nature of the notation.

Finally, although the editor was developed in a research environment with powerful equipment, some artificial restrictions were imposed during its development. This was done with the hope that porting this system to a workstation environment in the future would be easier. One such restriction was the decision to use only eight of the 32 available bit planes for the image displayed on the screen. This restricted, for instance, the number of static menu options available and the number of colours that could be displayed on the screen (although the second was not an issue here).

3 Editor Description

In this chapter we examine some implementation aspects of the Benesh editor. We begin with a brief description of the hardware and software environment in which development of the editor has taken place. This is followed by a presentation of the data structures employed to store the information required to reproduce a score, as well as a discussion of certain aspects of the design implementation. A description of a few examples of the user interface are then provided. Finally, we examine a mechanism developed to produce a score in hardcopy form.

3.1 Current Environment

The editor runs on a DEC VAX 11/780 minicomputer. It uses an Ikonas RDS 3000 graphics system attached to a colour display monitor on which most of the output and feedback from the program are shown. A Summagraphics Bit Pad tablet with puck is used as the primary input device for interaction with the program. Finally, a terminal is used to start the program, for entry of text strings, to display error messages and to display any 'help' information requested by the user.

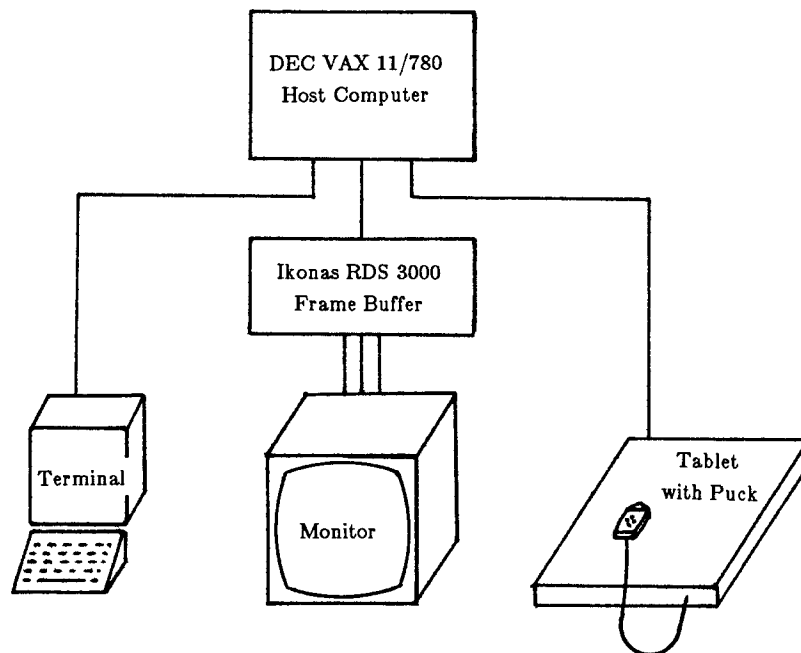


Figure 3.1 The Hardware Configuration

The Ikonas graphics system consists of a block of memory called the frame buffer in which the image to be displayed is stored, a video generation system that displays the image 30 times per second, an interface to the host computer, and a dedicated bit-slice microprocessor coupled to the frame buffer.

The *frame buffer memory* is configured to display 512 scanlines, each consisting of 512 pixels. Each pixel may contain up to 32 bits of information. As has been previously discussed, only 8 bits are used in this editor.

The *video generation system*, illustrated in Figure 3.2, consists of the frame buffer controller, a crossbar and colour lookup tables. The *frame buffer controller* reads the appropriate data for each pixel from the frame buffer and passes the information on to the crossbar. The *crossbar* is a programmable switch which allows the programmer to specify which of the 32 bits received from the frame buffer controller are connected to each of the 24 output lines. In a typical configuration, eight bits are used for red, eight bits are used for green, eight bits are used for blue, and the final eight bits are used for overlay. Output from the crossbar is in turn used to access an entry in each of three *colour look-up tables* (red, green and blue). The corresponding ten bit entry in the colour look-up table is converted to an analogue voltage which controls the red, green or blue electron beam to produce a pixel of the desired colour on the monitor.

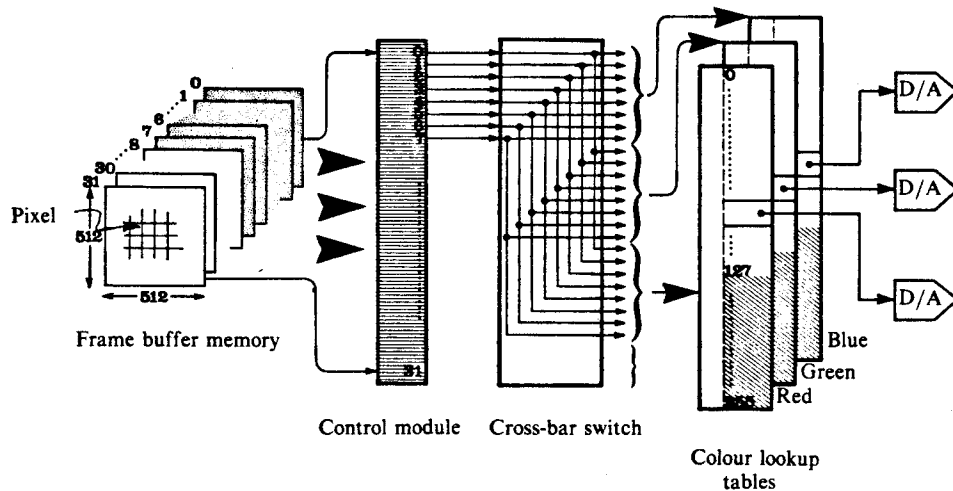


Figure 3.2 Video Generation System

The editor uses only eight of the 32 available bit planes. These eight bit planes are divided into two sets, which we will call the *foreground* and *background*. The seven background planes are used to store such static information as the stave lines and the notation information they contain, the various static menus and the background of the work-frame (see Figure 3.3). Each item (e.g. individual menu options) is written into the frame buffer memory using a different eight bit colour value. In this way each item can then be displayed in any colour (including black to make it invisible) independently by manipulating colour look-up table entries. This technique is commonly known as colour table animation [Shoup79, Booth82].

Interactive manipulation is done through the use of a package developed at the University of Waterloo by Paul Breslin [Breslin82]. This system is organised around a segmented display list and runs on the microprogrammable *bit-slice microprocessor* coupled to the frame buffer. The display list contains the graphical representation of dynamic information (such as the cursor, pop-up menus, Benesh signs and curves) that appears in the foreground plane. Each item is described in a *segment* which can be created, picked, modified, set visible or invisible, and deleted.

As interaction with the editor occurs, the frame buffer *write mask* is set so that only data stored in the foreground plane is manipulated. The microprocessor performs an *auto-clear* of the foreground plane during each display cycle. That is, the data stored in the display list is redrawn thirty times per second. This facility provides the effect of dragging a sign if its position changes from one display cycle to the next. This technique is used for cursor tracking and to position Benesh signs around the screen.

The editor and segmented display list system are supported by a graphics package also developed at the University of Waterloo and outlined in [Lea83]. This package provides the primitives needed to access the Ikonas system and input tablet, to produce text for display in the score and to produce the various Benesh signs and curves.

Most interaction with the editor takes place through the use of a graphics tablet and puck. The horizontal and vertical position of the puck on the tablet is determined and a *tracker symbol* or *cursor* is displayed at the corresponding position on the monitor screen. As the puck is moved on the tablet, the tracker also moves to reflect the new position. Interaction is effected, for instance, by

placing the tracker over a desired menu option or work-frame position (by moving the puck), and by pressing one of the four puck buttons to signal a request for action to the editor program.

Finally, the entire system runs on a DEC VAX 11/780 minicomputer running the UNIX operating system. All code for the editor is written in the C programming language.

3.2 Screen Layout

Let us now turn to examine the editor itself in some detail. We begin by looking at the general layout of the display screen while the editor is running as illustrated in Figure 3.3.

Two stave lines of the score currently being edited are shown at the top of the screen. A scroll bar indicating the position of these two lines in the entire score is visible to the right. A work-frame area is shown in the centre of the lower part of the screen. This work-frame is four times the display-size of a frame in the stave lines to allow more accurate manipulation of signs and curves in a frame. To the left of the work-frame a stave-menu is shown. This menu contains options that facilitate manipulation of the entire score (Quit, Archive and De-Archive) as well as manipulation of the displayed stave lines (Scroll lines, Add, Edit, Delete, Move frame(s), etc.). If the user wishes to Add or Edit a frame, the frame-menu and body-menu appear. These are made up of options that facilitate the manipulation of individual frames (specify, modify and delete signs and curves). To the right of the body-menu are two work areas used for the specification of certain Benesh signs.

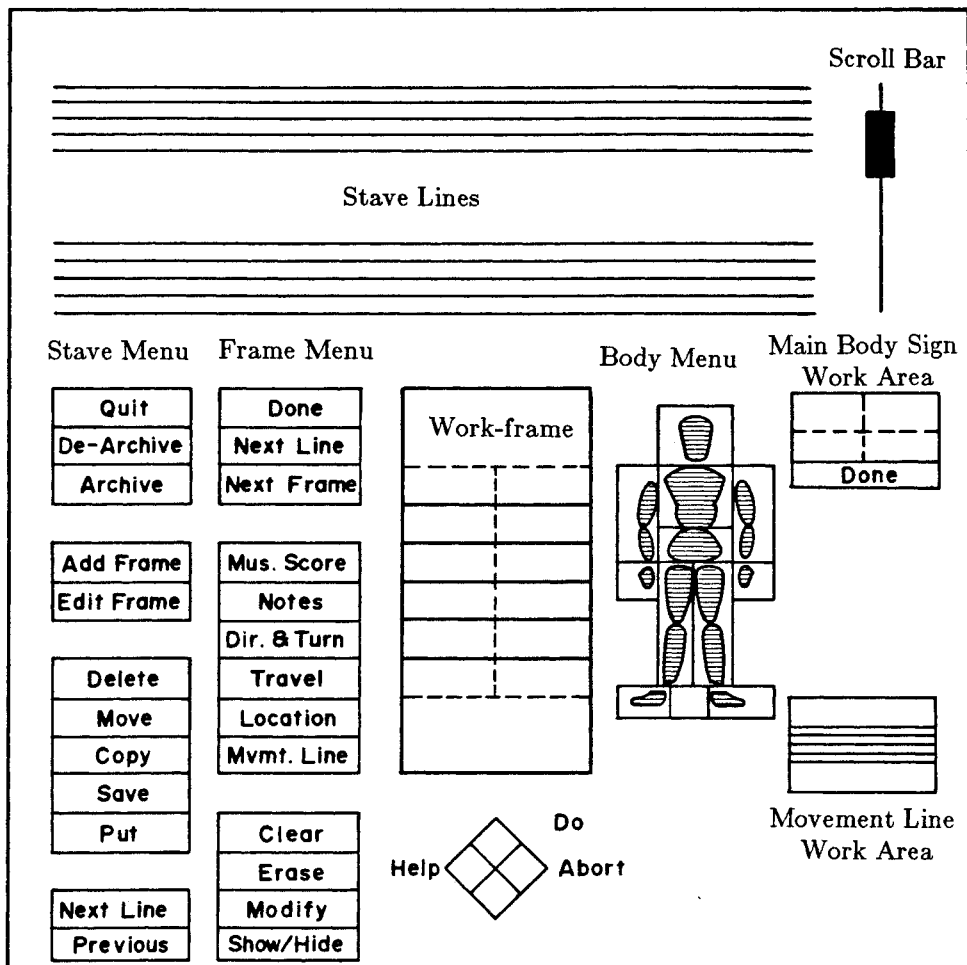


Figure 3.3 The Screen Layout

This is all the information stored in the background planes. Note that only those menu items selectable (or selected) and any work area in use at any given moment are actually visible.

3.3 Data Structures

Let us now examine the data structures employed to represent the basic elements of the score. A Benesh score consists of one or more stave lines, each containing one or more frames. Each frame may contain header, body-data and/or scoring information. The information and attributes maintained to record details about a frame are described, followed by an examination of how these details are represented in the editor.

3.3.1 The Score: Stave Lines and Frames

In the earlier version of the editor, the score was represented using a doubly linked list of frame nodes, each corresponding to a frame in the score. To accommodate the notion of score lines in the version of the editor described here, a second doubly linked list has been employed (see Figure 3.4). This is a list of the *line nodes*. Each line node corresponds to a stave line of the score and has a pointer to its predecessor, a pointer to its successor and a pointer to the first frame on the line.

In addition to the data stored in the *frame node* in the earlier version of the editor:

- a pointer to its predecessor,
- a pointer to its successor, and
- a list of Benesh signs defined in this particular frame,

the following information is now also maintained for each frame:

- a description of movement lines active in this frame,
- a frame-type,
- the left-most extent of a sign or curve and the right-most extent of a sign or curve in the frame, and
- the position (offset) of this frame on the stave line of which it is a part.

Each frame may contain any number of signs and all or part of any number of curves. *Benesh signs* to define various types of information are placed in the frame. In some cases the position of a particular type of sign is predefined by

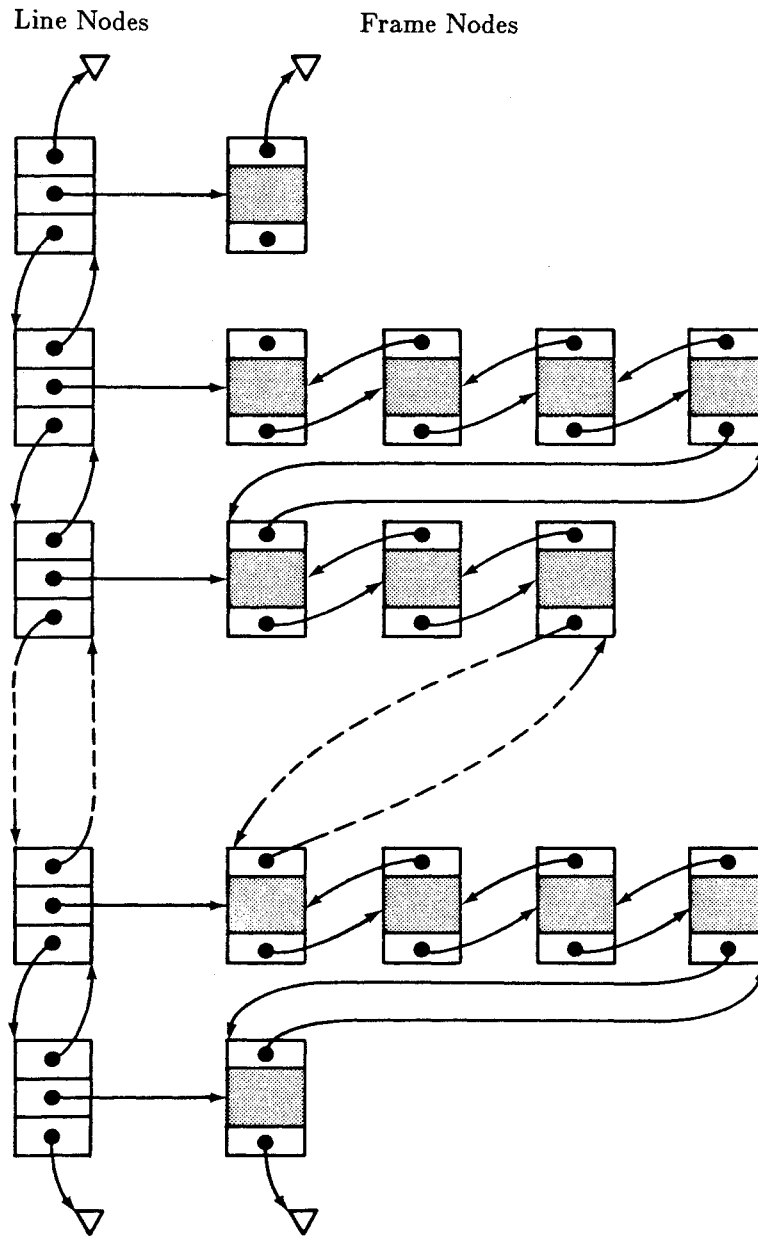


Figure 3.4 Stave Lines and Frames

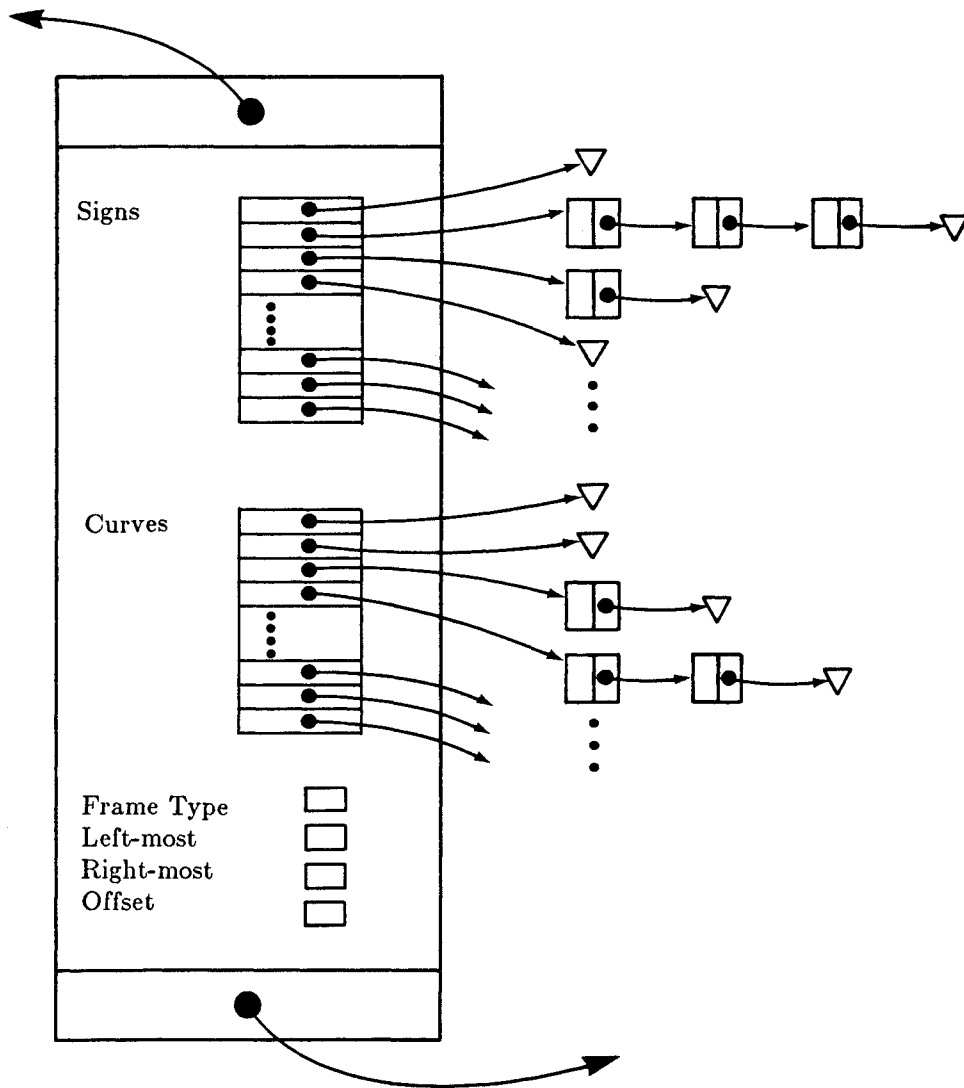


Figure 3.5 A Frame Node

the rules of the notation. In other cases a sign may be placed in the frame by the user in order to associate it with another sign or to allow space for other information appearing in the frame. The coordinates for a frame run from zero to five in the x-direction and from zero to ten in the y-direction. Specific signs are used to define the following information:

- time signature to indicate the number of beats in each bar,
- note – a string of text used for titles, musical terms or a brief written description of something not expressed directly in the notation,
- direction faced on stage,
- turn,
- travel,
- stage location,
- rhythm indicators to specify a beat which may be sub-divided into parts,
- body part position (for example, to specify position of the head, arm, knee, feet, etc.),
- hop or bounce movement,
- bar lines to mark the beginning and end of each musical bar or the end of a sequence, and
- repeat signs to indicate that a given sequence of positions and movements is to be repeated.

A frame may also contain numerous *curves* to express transitions between positions. Such movement lines are divided into two main types;

- (i) Free-Hand lines are those ‘drawn’ by the user. These are a free-form line whose starting and ending points are located within a single frame.
- (ii) Locomotion lines indicate motion such as a step, jump or slide. The editor creates the indicated line according to some general rules to approximate the predefined shape for the particular type of line. The shape of this curve may be modified by the user. Each line usually starts in one frame and ends in a subsequent frame but it may, in some cases, be contained entirely in a single frame.

The end of each movement line is *attached* to or ends under a Benesh sign in the frame.

The *frame-type* is determined by the information contained in the frame. Various frame types are necessary in order to facilitate the implementation of a spacing algorithm for displaying the frames in a stave line. The following frame types are defined;

- (i) A Header Frame is found at the beginning of a score sequence. It contains a time signature and possibly a text string to describe or title the score.
- (ii) A Scoring Frame contains a bar line or repeat sign. This type of frame may also define direction, turn, travel, or location information and may contain a note. It may not contain a time signature, the definition of any body part or a rhythm sign.
- (iii) A Body-Data Frame contains the definition of any body part or a rhythm sign. Such a frame may also contain direction, turn, travel, and location signs and a note. It may not contain time signature, bar line or repeat signs.
- (iv) A Don't-Know Frame: It is clear that a frame may contain no data at all or it may contain direction, turn, travel and location signs and a note and it would not fit into one of the above categories. Such a frame is known as a 'don't-know' frame. It is manipulated by the editor in the same manner as a scoring frame. Once a time signature, bar line, repeat, rhythm or body part is defined in such a frame, its frame-type changes to the appropriate one of the previous three.

3.3.2 The Frame: Signs and Movement Lines

For the following discussion it will be necessary to distinguish between the terms sign and symbol. These terms have been used in different ways by different writers. The vast majority of published material dealing with Benesh Movement Notation uses the term *sign* to identify those elements of the notation used to define a body-part, direction, travel, etc. [Benesh56, Benesh77, Ryman82, McGuinness-Scott83, Brown84, Ryman84], and we will use it in this way here. We now also define *symbol* to describe an instantiation in the editor of all or part of a Benesh sign; that is to say, such items as horizontal and vertical strokes, circles, etc.

Due to the plethora of Benesh signs, to predefine each sign as a symbol in the editor would not be feasible. Thus, each sign is defined in one of three ways, either (i) as a single predefined symbol, (ii) as a combination of basic predefined symbols or (iii) as a combination of straight lines. In some cases signs are constructed by the user from a number of predefined symbols. One such example is the creation of a repeat sign. One repeat sign consists of two straight lines and possibly one or two qualification symbols. Each of these sub-parts of the repeat sign is a basic symbol and the entire sign consists of a particular combination of such symbols. In another case, the main body signs (representing head, torso and pelvis) are created by the user by defining a combination of two straight lines (where each line consists of an instruction to move to a start position and a second instruction to draw to an end position). In the case of a main body sign, there are three types of displacement (tilt side to side, turn left to right and bend backward to forward). Each has seven possible positions in the notation, resulting in a total of 343 signs.

As the editor is initiated, and after opening the input and output devices required, an internal library of predefined symbols is created. This is done by reading a file containing instructions as to how each symbol is to be constructed. These primitives include: define a symbol name, move, draw, circle, text and call another symbol. A segment is created for each symbol using the display list package and the instructions used to create it are also stored internally to facilitate later manipulation (e.g. rotation) of the symbol. Each symbol is now uniquely identified by a segment name.

The list of Benesh signs defined in a particular frame is stored by specifying the sign and its position in the frame. If the item being described is defined by a single symbol or by a combination of symbols, a linked list is created as illustrated in Figure 3.5, the first element of which is pointed to by the appropriate element of the sign array. Each node in the list defines:

- a segment name identifying the symbol,
- an (x,y) position of the symbol in the frame, and
- the degree of rotation of the symbol.

Similarly, if the item is defined by a combination of line segments, a linked list of line elements is created and pointed to by the appropriate element of the sign array. Each of these nodes contains:

- the starting point (x,y) , and
- the ending point (x,y) of the line segment.

Finally, an element of the sign array for the item in question may point to a node defining a text string. In this case the node contains:

- the starting position of the string (x,y) , and
- a copy of the text string to appear in the frame.

Curved lines are used in Benesh notation to represent movement and can take on an infinite number of shapes and lengths. In fact, the length of one particular line may vary depending on changes made to the rest of the stave line on which it appears. The general shape of a step, jump or slide line is defined by the notation and approximated by the editor. This shape may be altered by the user in some cases. Free-hand drawn lines may take any shape desired by the user. As a result of this high degree of variability, it was decided to render movement lines using B-splines [Bartels83]. As a result, the storage representation of a curve is that of a series of control vertices (see the Some Implementation Details section of this chapter). That is to say, for each curve from four to twenty points (control vertices) are stored.

In the case of free-hand drawn curves, the entire curve is contained within a single frame. The control vertices are stored directly in the coordinate system of a frame. In the case of movement lines that begin in one frame and end in another this is not possible. Consider, for instance, a sequence of frames on a stave line labelled Frame A through Frame H and a step line beginning in Frame D and ending in Frame F (Figure 3.6).

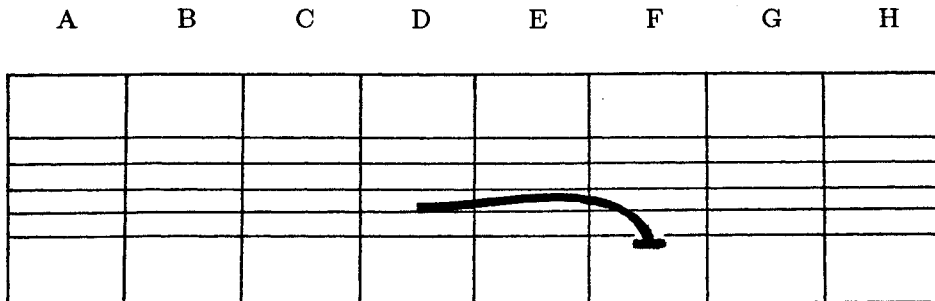


Figure 3.6 Locomotion Movement Lines

If a frame is added to, deleted from or modified on this stave line, the positions of each frame (and thus its contents) change relative to the beginning of the stave line. The step line in question must retain its relative starting position within Frame D, its relative ending position within Frame F and its shape. Relative to the stave line however, its starting position, ending position and length must be changeable. To accommodate this variability, curves are defined in *normal coordinates* [Foley82] in the x-dimension. That is to say, such curves are defined in a coordinate space ranging from zero to one (Figure 3.7). When the editor draws a locomotion line on the stave line, the curve is calculated in this coordinate system, translated to begin at the correct position on the stave line and scaled to end at the correct position. The y-coordinates of the control vertices are recorded in the world coordinates used in the lines and frames and do not require any translation or scaling when displayed.

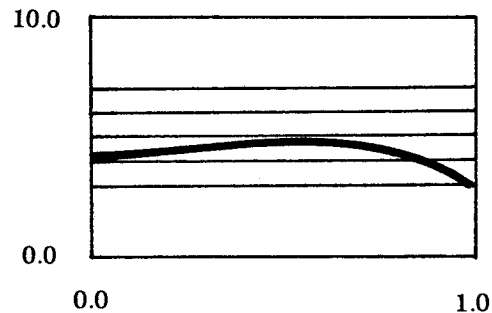


Figure 3.7 Locomotion Line Coordinates

A second array of pointers (the *line array*) is stored in each frame to reference structures describing *active* curves in that frame. A curve is considered to be active in a frame if (i) it is contained entirely in the frame, (ii) it begins or ends in the frame, or (iii) it begins in a preceding frame and ends in a subsequent frame. The end of every movement line is attached to/positioned near and associated with a Benesh sign in a frame. A linked list of active movement lines associated with an item is referenced by the corresponding entry of the line array. Each node in the linked list provides data about a free-hand drawn curve or a locomotion curve. For a free-hand drawn curve, the node consists of:

- the number of control vertices used to define the curve,
- a pointer to the list of control vertices,
- the left-most and right-most extent of the final line within the frame,
- the type of qualification associated with this line (if any), and
- the (x,y) position of that qualification.

For locomotion curves the information stored varies depending on what part of the line is active in the frame in question. If a locomotion line is contained entirely in the given frame, it is known as a contracted movement line and the information stored is:

- the number of control vertices used to define the curve,
- a pointer to the list of control vertices,
- the starting and ending points of the line in the frame (recall that movement lines are stored in normal coordinates in the x-dimension and thus actual starting and ending points must be stored explicitly in order to perform the translation and scaling operations when the line is rendered),
- a qualification associated with the line (if any), and
- whether or not the line is qualified as a 'skimmed' movement (for jumps and slides).

In the case of a movement line which spans two or more frames, as illustrated in Figure 3.6, the data stored locally in each frame depends on whether the line *begins* in the frame (as in Frame D), is *in progress* in the frame (as in Frame E) or *ends* in the frame (as in Frame F). If such a movement line begins in a given frame, the information stored in that frame is:

- the number of control vertices used to define the curve,
- a pointer to the list of control vertices,
- the position in the frame (x,y) at which the line starts,
- a pointer to the frame in which the line ends (in our example, Frame F),
- a pointer to the corresponding 'end-line' node in the end-frame,
- the position of the end of the line (x,y) in the end-frame,
- the qualification associated with this movement line (if any), and
- whether or not the line is to be rendered as a 'skimmed' movement (for jumps, hops, bounces and slides).

In frames in which a locomotion line is in progress (in our example, Frame E) or in which the line ends (Frame F), the information stored is:

- a pointer to the frame in which the line begins, and
- a pointer to the 'start-line' node in the start-frame.

3.4 Some Implementation Details

Now that we understand what information is maintained and manipulated by the editor, an explanation of design decisions made and their implementation is necessary for a more complete understanding of the operation of the editor. The following is a description of some features and functions that have been developed in this version of the editor. A discussion of the underlying reasons for the design decisions may be found in the next chapter.

3.4.1 Score Formatting

As discussed earlier, the system being described here is both an editor and a formatter. When a score is produced manually by a choreologist, the frames are spaced within bars and bars on stave lines to provide visual clarity. This is a complex task to emulate perfectly in the editor, which indeed implements only a few simple spacing rules. Additional study is needed in this area to provide a more sophisticated algorithm for the spacing of frames on a stave line.

For the purpose of line spacing, two values are calculated for each frame. The first is the x-value of the position of the left-most sign, the left-most extent of a free-hand drawn line, or the **beginning** of a locomotion line. The other is the x-value of the position of the right-most sign, right-most extent of a free-hand drawn line, or the **end** of a locomotion line. The maximum distance from the centre of the frame to one of these two values is doubled to obtain the *data width* of the frame. Thus, the data width is the sub-section of the frame (in the x-direction), centred on the frame centre line, in which all signs and in-frame curves are located. This is done to retain the perceived symmetry of the frame about the centre line. For example, in a frame in which at least one sign is located near the left or right edge, the data width will be near five. On the other hand, in a frame in which all signs and in-frame curves occupy a small band near the body mid-line, the data width may have a value near one. When a stave line is to be displayed, the data widths of all the frames on the line are summed. This value is subtracted from the stave line length to give us the amount of total *inter-frame space* available. This space is distributed proportionally between frames as follows:

- one unit at the beginning of the stave line and one unit at the end of the stave line,
- two units between any two adjacent body-data type frames, and
- one unit between all other frames (i.e. between a header, scoring or don't-know frame and any other frame).

One complication arises when the sum of the data widths of the frames on a line exceeds the stave line length. A number of actions could be taken when this occurs. One could *push* the last frame from the end of the overfilled stave line onto the beginning of the next stave line. This could, in turn, cause that line to *overflow*. This process would go on until enough room is found on a line for the overflow of the previous line or until the end of the score is reached and a new stave line is added. This is undesirable for two reasons.

First, the frames pushed off onto the next line may 'split up' a bar, leaving the first frame(s) of a bar at the end of one stave line and the remaining frames of the bar at the beginning of the next line. Whenever possible, choreologists prefer to move the entire bar from the end of an overflowed stave line onto the beginning of the next line and respace the frames remaining on the original line. Wherever feasible, this is also done by the editor. The choreologist may choose to override this and manually split a bar using stave edit operations such as the Move or, while the score is being entered, the Next Line command.

The other problem that arises is the possibility that this overflow process could affect a large number of stave lines (possibly, from the current position to the end of the score). This is often undesirable when a score is near completion and the choreologist is happy with the spacing of the score. If a spacing modification is necessary we would like any spacing effects to remain local and not affect the rest of the score. Hence, if an overflow occurs, a new stave line is created and inserted into the score after the line that has overflowed. The overflow frames are placed on this new line. If necessary, the score is scrolled so that the new stave line appears as the second displayed score line. As a result, any spacing changes remain local and the user may then rearrange the spacing, through the use of stave line operations, such as Move, Copy and Delete frames, in the vicinity of the overflow. Thus, the spacing of the rest of the score remains unchanged.

3.4.2 Determining Frame-Types

As we have seen earlier in this chapter, a number of frame types are distinguished in order to facilitate the implementation of certain functions of the editor. The frame-type is determined automatically by the existence (or absence) of data defining specific items in the frame as previously outlined.

In the process of creating a score, the normal sequence used is to execute an 'Add Frame' operation followed by numerous 'Next Frame' or 'Next Line' operations and finally the 'Done' operation. Each Next Frame or Next Line command appends (or inserts) the work-frame into the score and the next work-frame appears. This work-frame is assumed to be a body-data frame. That is to say, certain information, such as direction, stage location, etc. is copied from the frame immediately preceding the position of the new frame in the score (scoring, body-data or don't-know type frame). In addition, body data is obtained from the most recent body-data frame (if any) and copied into the work-frame. Thus, the work-frame is, by default, a body-data frame containing the most recent accumulated data. If the user wishes to enter a bar line or repeat sign at the current place in the score, it is necessary to perform a 'Clear' operation first. (Recall that a body-data frame may not contain a bar line or repeat sign.) Clearing the frame causes it to become a don't-know frame and defining a bar line or repeat sign is then possible.

3.4.3 The Percolation of Changes

We have seen that the editor, although it shows only those Benesh signs that indicate a change, maintains cumulative information in frames. This fact must be taken into consideration when inserting or deleting a frame in the score.

For the purposes of the following discussion, we will use the specification of the head position as an example. Consider the case in which a frame that redefines a head position is deleted. In the subsequent frame, the head is defined with the same position as in the deleted frame. The choice now is to leave this frame with the head position just deleted from its predecessor or to replace the head definition with that found in the frame preceding the deleted frame. The second approach is taken in the editor. As a result, the editor must scan forward through the score from the modified frame, replacing each definition of the head equivalent to the original with the definition of the head found in the frame preceding the deleted frame. This operation continues until

the head position is redefined or the end of the score is reached.

Similar considerations arise when inserting a frame. In this case, the frame preceding the insert position and the one succeeding it usually record the same head position. If a frame which redefines the head position is inserted, the editor again scans forward from the insert position through the score replacing each definition of the head equivalent to the original with the definition found in the inserted frame until the head position is again redefined or the end of the score is reached. In addition, when inserting a frame, if any items corresponding to this frame type are not defined, they are copied from the frame(s) immediately preceding the insert position.

Similar operations are performed when an entire sequence of frames are inserted or deleted using the Move, Copy, Delete, and Put operations.

It should also be noted that the percolation of these changes must take the frame type into consideration. Direction and location data are percolated through scoring, body-data and don't-know type frames. The definition of a body part is percolated only through body-data type frames (scoring and don't-know frames are ignored).

Finally, it is semantically incorrect to percolate certain signs as described above, namely any time signature, rhythm, turn, and travel sign as well as any note defined in the frame.

3.4.4 Score Edit Operations and Movement Lines

With the introduction of movement lines, an interesting issue arises when some score edit operations (Move, Copy, Delete, Save and Put) are performed. These operations make use of an internal buffer called a *save-buffer* in their implementation. This buffer is independent of the actual score itself. It simply contains a copy of an arbitrary sequence of frames from the score.

Let us first consider the Save operation. The user is asked to define the sequence of frames to be copied to the save-buffer. The signs contained in each frame are copied to a corresponding frame in the save-buffer. Now, if a movement line is contained entirely within the sequence of frames indicated by the user, it can also be copied into the save-buffer. If however, a movement line starts in a frame prior to the beginning of the indicated sequence and ends in a frame within the sequence, it is not copied into the save-buffer; incomplete line specifications are ignored during the copy operation. The same is also true in the

case of a movement line which begins within the chosen sequence and ends in a frame after the end of the sequence. If a movement line starts in a frame prior to the beginning of the indicated sequence and ends in a frame after the end of the sequence, it is also ignored during the copy operation into the save-buffer. Since Delete, Move and Copy all use the Save operation as part of their implementation, this discussion also applies to these operations. As a result, when the save-buffer is inserted into the score using the Copy, Move, or Put operations, no incomplete lines are inserted into the score.

Thus, in the case of a Delete operation, a movement line that begins within the deleted sequence and ends outside this sequence (or vice versa) is deleted in its entirety from the score. Lines which begin prior to the sequence and end afterwards are simply 'shortened'.

One situation in which this property is disconcerting occurs when a few frames are Moved from the end of one stave line onto the beginning of the next, or vice versa, and these frames contain the beginning or ending of movement lines. Because the Move operation is composed of a Delete and a Put, the indicated frames are first copied into the save-buffer (ignoring incomplete movement lines), then deleted from the score (removing any incomplete movement lines) and finally, the contents of the save-buffer are copied back into the score. Although the sequence of frames has been removed and re-inserted in the same position relative to the other frames of the score, the movement lines in question have been deleted and must be redefined.

3.4.5 Rendering Movement Lines

A series of interesting problems were addressed prior to and during the implementation of movement lines. A number of prerequisites were defined for this implementation. These included:

- an efficient storage representation,
- the facility to manipulate (reshape) a curve easily,
- the ability to 'smooth out' a curve, as well as
- the capability to accurately render complex curves (such as one containing two loops).

After some experimentation, it was decided to define, manipulate and render curved lines using B-splines [Bartels83].

This class of splines is defined by a sequence of *control vertices* forming a *control polygon* near which the curve passes. The curve is represented using piecewise polynomial parametric functions. A sequence of polynomial segments are joined smoothly at their end points to produce the complete curve. Currently, the editor uses 29 chords to render a movement line, regardless of the number of polynomial segments it consists of.

Knots in the parameter space are values of the parameter corresponding to an end point of the curve or to one of the joints at which two segments meet. The knots are uniformly spaced in the parameter space, with the exception of the two endpoints of the curve. Here the knots have a *multiplicity* of four to ensure endpoint interpolation.

Using distinct knots and control vertices defines a curve with a least second degree continuity. For most movement lines this is desirable. In one case however, we force the same knot value to correspond to three interior control vertices (knot multiplicity of three) reducing the degree of continuity at that point in the curve. This is necessary to render certain slide lines. In Figure 3.8 we have assigned the knots at points B, C and D to three control vertices each to provide the necessary sharp change in direction at these points on the curve.

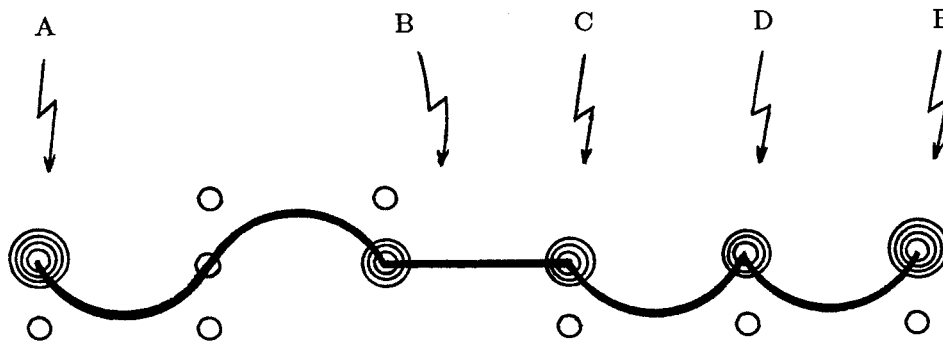


Figure 3.8 Rendering a Slide Line

For each curve representing a movement line, end point interpolation is forced (points A and E) in order to ensure the end points of the curve coincide with the end points of the desired line.

A key aspect of B-splines, and one that plays an important role in the Benesh editor, is the property of *local control*. Changing the position of a control vertex only affects a small neighbourhood of the curve (in general, four adjacent segments of the curve). This property is exploited extensively in the process of modifying a generated curve.

Using the B-spline technique, a curve in a Benesh score may be represented by as few as four control vertices or as many as 20 control vertices (with the exception of slide lines; here the number of control vertices used is determined by the complexity of the line). The number used may be specified by the user. Using more control vertices makes the neighbourhood that each affects smaller. The effect of varying the number of control vertices and of moving a control vertex are discussed elsewhere in this chapter.

3.5 Some Interaction Details

We now turn our attention to the editor's user interface. Although minor modifications were made to the functions available in the earlier version of the editor, the general mode of interaction was retained. These interaction techniques were also applied to new functions in the current editor. Where new interaction techniques were required, they were designed to be consistent with those developed earlier. A brief description of the actions the user performs to manipulate stave lines and frames is now available on-line from within the editor in the form of Help menus (see Appendix C). In the following section, we present some of the modifications made to the original user interface as well as some extensions made to accommodate new types of interaction.

A number of techniques are used in order to specify a Benesh sign. These include (i) selection and placement of a simple sign, (ii) selection and placement of a number of symbols that together form a sign, and (iii) other methods to build a sign. We shall look at an example that typifies each of these as well as the specification and manipulation of a curve.

3.5.1 A Single Symbol Sign

There are a number of Benesh signs that consist of a single symbol. In such cases a pop-up symbol menu showing the available options is displayed. This pop-up menu usually appears in the vicinity of the current tracker position in order to minimise eye and hand movement. In some cases, the menu is displayed in a fixed position on the screen where it is not cluttered by other information. One advantage to this method is that the user knows where to expect the menu to appear and may move the tracker toward or to the position of the symbol to be picked even before it is visible [Foley84]. On selection of an item from this menu, the symbol is placed in the work-frame. Placement of a sign is either done automatically, if the notation predefines the position, or the sign is dragged into the work-frame by the user. Examples of signs for which the notation predefines the position are Time Signature, Rhythm, Travel and Direction. On the other hand, if the final display position of a sign may be determined by the notator in order to provide semantic information or to allow space for other information in the frame, the tracker changes its shape to that of

the selected symbol and is dragged into the work-frame by movement of the puck and placed by pressing the yellow (top) puck button. Let us examine the process of defining a body limb in order to illustrate this type of interaction.

The definition of a body limb consists of selecting a symbol for the bend (elbow or knee) and/or a symbol for the extremity (hand or foot). When an arm or leg is chosen from the Body-Part menu, a pop-up menu of all possible symbols for defining the bend and the extremity (including basic signs and contact signs) is displayed. The user selects one of the available symbols. If a bend symbol is chosen, the tracker becomes that sign and is dragged into and placed in the work-frame. The bend symbols of the pop-up menu disappear and selection of an extremity symbol may now take place. In addition, a 'Done' option appears to the right of the work-frame near the position at which the bend has been placed. Although the Done option is part of the resulting pop-up menu, it is placed near the work-frame in order to minimise hand movement should the user not wish to further define an extremity. Similarly, if an extremity symbol were picked, after it is dragged into and placed in the work-frame, the bend symbols remain available for selection, as does the 'Done' option described above. After selection of both a bend symbol and an extremity symbol, or of the 'Done' option, the user is returned to the Frame Edit and Body-Data menus.

3.5.2 A Multiple Symbol Sign

There are a few examples in the editor in which the user must construct a single Benesh sign from a number of individual symbols. These include the repeat, turn and location signs. As an example of this type of process, let us look at the generation of a stage location sign.

A stage location is specified by selecting a symbol indicating the position between stage-left and stage-right and placing a small cross symbol on this symbol to illustrate the position between front-stage (downstage) to back-stage (upstage). There are a large number of possible location signs. First, one of nine location symbols (from stage-left to centre stage to stage-right) is chosen (see Figure 3.9). A cross-bar appears below the chosen symbol to provide feedback. Now, if the location to be defined is between two of the nine possibilities, the adjacent location symbol must also become part of the sign. It may now be selected. The menu disappears and the composite symbol selected appears. The tracker now becomes the cross-bar and the user places it on the displayed symbol where desired. On the other hand, if a second left-right symbol

is not required, the cross-bar displayed below the choice is picked. The menu disappears, the chosen symbol is displayed and the tracker becomes the cross-bar. It is then placed on the symbol to form the required sign. When the sign has been completed, the tracker then becomes this sign. The user drags it into the work-frame and places it in the required position.

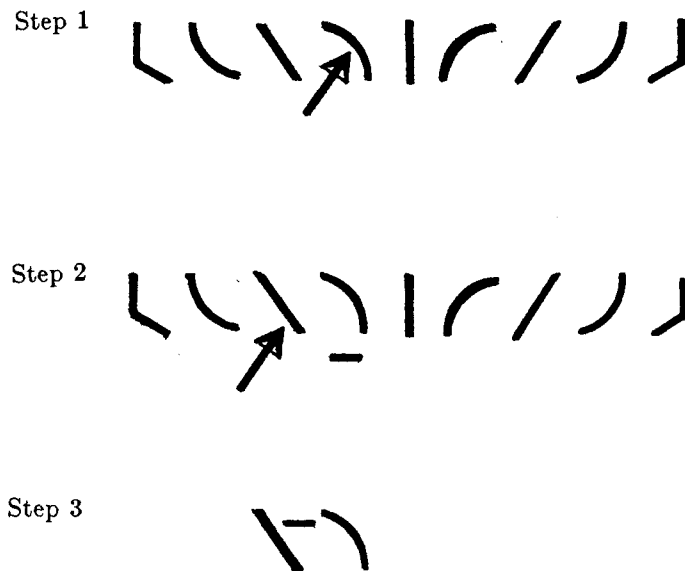


Figure 3.9 Defining a Stage Location

3.5.3 Other Methods of Building a Sign

The above discussion has dealt with signs that are constructed of one or more predrawn symbols. This approach is possible when there are a limited number of signs involved. There are some cases in which the number of possible signs is too large to employ one of the user interface techniques discussed above. One such example is that of a main body sign (head, torso or pelvis) and the technique used to define the sign is described in detail in [Singh82a]. The interface has since changed slightly in that continuous tilt, bend and turn are no longer allowed. Rather, seven discrete positions are predefined for each. Also, dashed lines have been superimposed on the work-area to provide more accurate feedback of the positions selected.

Turn signs are another example in which a sign is constructed. A turn may begin with a dancer facing any one of eight directions faced on stage, may be in a clockwise or counter-clockwise direction and may end in any one of 15 directions ($1/8, 1/4, \dots, 7/8, 1, 1-1/8, \dots, 1-7/8$), resulting in 240 possibilities. In addition, some of the above signs may be qualified with a '2', '3' or 'x' to further specify the number of turns, resulting in over 600 possible signs to define a turn. In the editor, we deal with this problem in the following manner. When the user indicates that a turn sign is to be defined, a pop-up menu of the eight basic stage directions faced appears. The user selects the direction symbol, indicating where the turn is to start. The tracker is then moved in a clockwise or counter-clockwise direction, corresponding to the direction of the turn, and stops at the direction symbol indicating where the turn is to end and selects this symbol. As the tracker is moved the editor provides feedback by drawing the path that the tracker follows (see Figure 3.10). If the turn sign defined in this way is one of those that may be further qualified, a menu of the four possible combinations of that turn sign and a qualification appears. The user is required to select the appropriate combination. When specification of the Turn sign is complete, the tracker becomes that turn sign and it is dragged into the work-frame and placed where required.

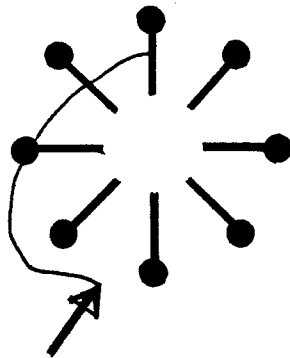


Figure 3.10 Defining a Turn

Note that the operation described here is similar to drawing a turn sign by hand. However, this interface allows the choreologist to think of a turn sign in terms of the movement being described (i.e. direction faced at start, direction of turn and direction faced at end) rather than in the terms of the notation symbol used (e.g. 1-3/8 spiral starting from the downstage direction). In this way the editor attempts to prevent the sign from being treated simply as a symbol [Benesh77].

3.5.4 A Locomotion Movement Line

The specification of locomotion movement lines differs from other analogous functions of the editor in that such a movement line generally begins in one frame and ends in another. This poses an interesting design question. Should the movement line (i) be treated as a single entity and specified in its entirety in a single operation, or (ii) be specified logically as the score is created; that is, the start of the movement line is specified in the frame in which it begins and the end of the line in the frame in which it ends? The first of these choices more closely reflects the way in which a choreologist would actually construct such a line. When the necessary signs have been placed in both the start-frame and the end-frame, the line is drawn appropriately. This is the approach taken in the Benesh editor developed on the Macintosh computer described elsewhere. The editor we are describing here, however, uses the second approach. The reason for this is addressed below. Let us consider the specification of a jump line to illustrate this approach.

Entering a Benesh score consists of the construction of one frame after the other in the work-frame area and then placing it in the score. As the user builds the frame in which a jump line is to begin, the 'Movement Line' option of the Frame Edit menu is selected. A pop-up menu appears requesting the type of movement line being specified to be indicated. The 'Start' option is chosen. The pop-up menu is then replaced by another menu allowing the user to indicate that a jump line (as opposed to a step or slide line) is to be started. The editor verifies that a foot or both feet sign has been specified in the work-frame because one cannot notate a jump without providing a description of the foot/feet position. A "Start Where?" icon replaces the pop-up menu, asking the user to specify the (x,y) position in the work-frame of the start of the jump line. The position indicated must lie *near* (within a single unit of) a Benesh sign in the x-dimension before it is accepted by the editor. A dashed line is drawn from the

start position to the right side of the work-frame to indicate that an unfinished movement line has been specified.

The user continues the editing process. Subsequent frames will now show a dashed line from the left side to the right side of the work-frame to indicate that an unfinished movement line exists.

When the user reaches the frame in which the jump line is to end, the 'Movement Line' option of the Frame Edit menu is again selected. The pop-up menu now contains an 'End Jump' option to allow the specification of the unfinished jump line to be completed. Again, the editor first ensures that a foot/feet sign has been specified in this frame, to which the jump line may be attached. The user is prompted with an "End Where?" icon and asked to select the (x,y) position in the frame where the line is to end. This must also be near a Benesh sign. A pop-up menu appears allowing the user to specify a qualification for the jump line being completed. The editor creates a jump line according to rules that approximate the general shape of such lines. The qualification, if any was chosen, is automatically positioned on the finished line. The dashed line in the work-frame is erased and replaced with a solid line approximating the movement line just specified. The entire line (with correct shape) is first shown when the work-frame is inserted onto a stave line.

One can see from this discussion that the specification of a movement line is treated at a more abstract level than defining such a line as one would draw it on paper. In some sense, the line is defined semantically. The editor provides some basic checking mechanisms to ensure that it is semantically correct.

3.6 Typeset Output

As work on this editor has progressed, a parallel effort was undertaken by Renata Kraszewski and John Chapman to provide hardcopy output of a Benesh score produced by the editor. The printed copy can then be used for archiving or publishing purposes. A separate program has been developed which reads as input a score produced by the editor and reproduces it on a number of devices, including a Versatec V-80 raster plotter (200 dots per inch), an Imagen/Canon 10/240 laser printer (240 dots per inch) and an Autologic APS-Micro5 typesetter (723 dots per inch). An example of each of these may be found in Appendices D, E and F respectively.

4 User Interface Issues

During the development of the work described in this report, the opportunity was taken to explore and test a number of human interaction issues. Human factors play an ever increasing and critical role in the acceptance and ease-of-use of any system today. The costs of poorly designed interfaces can include degraded user productivity, user frustration, increased training costs, and the need to redesign and re-implement the user interface [Foley84]. As a result, a great deal of emphasis is being placed on many individual aspects of the design and implementation of user interfaces in current research. Topics of recent studies include:

- ergonomic factors in the design of hardware to eliminate such ailments as eyestrain, back problems, etc.;
- use of appropriate input devices for various types of operations;
- how individual factors affect user productivity; this area includes such varied topics as user-initiated versus system-initiated dialogue, simplicity and consistency of the interface, demands on short-term and long-term memory, use of colour, effect of computer response times on user comprehension, appropriate use of feedback, etc.;
- techniques used to increase speed of learning a system (e.g. the availability of a help system, use of on-line tutorials, etc.).

Let us first define what is meant by the term *human interface*. Foley has suggested the following:

- the definition (including format) of all inputs from the user to the system,
- the definition of all outputs from the system to the user, and
- the definition of the sequencing of the inputs and the outputs [Foley82a].

In a more recent article, Foley et al. place greater emphasis on the role of the user in the design of human interfaces: “Interaction with a computer involves three types of basic human processes: perception, cognition, and motor activity. The system designer’s job is to design interaction techniques which minimise the work required by these processes, both individually and in combination” [Foley84].

He goes on to explain: “Perception is the process whereby unintelligible physical stimuli (generated in this case by the computer) are received by the receptor organs, transmitted to the brain, and are there recognised by a process theorised to be akin to pattern recognition.” Clearly, in the computer graphics environment, this process is influenced primarily by the presentation of information on the display device. Attempts to minimise the efforts required in the perception process affect the design of the user interface so that the user can quickly locate and recognise the items of interest. This has influenced such aspects as the use of colour, the choice of static or pop-up menus, and the choice of text or icons to represent options in a menu.

“Cognitive psychology deals with how we acquire, organise, and retrieve information.” This is becoming an increasingly important aspect of the design of the user interface. Research in this area provides insights into such things as ways to structure hierarchical menus, the number of choices to present, the types of words (or icons) to use, etc. Studies indicate that designing a system which fits into categories or concepts the user already understands, improves learning speed and productivity when using the system. Knowledge of this area aids in the development of a user’s conceptual model of the system.

“The motor process comes into play when the user, having received, recognised, and decided how to respond to the stimuli, performs a response in physical actions.” In *ChoreoScribe*, this usually involves moving the puck on the tablet. In some cases, however, the required response is to enter a string at the keyboard. One may also consider the location of the centre of attention (placement of menus, for example) to influence this process as well.

All these factors are taken into account during task analysis (or requirements specification). They also influence decisions made during the user interface design phase. The result is the definition of an abstraction of the system presented to the user.

The design of the user interface, in turn, influences the overall design of the software. In addition to the purely human factors issues described above, other facets of software engineering are affected when designing such a highly interactive system. Some software engineering issues that may arise include:

- the use of User Interface Management Systems (UIMS),
- interface prototyping,
- modularity and layering techniques,
- timing considerations with respect to feedback, and
- adaptable user interfaces (i.e. the capability to modify the user interface for individual users or groups of users either automatically or explicitly by the user).

This chapter deals with a number of issues that have influenced both the design of the user interface and the implementation of ChoreoScribe. Examples are used to illustrate each issue and, in some cases, a particular example may also be used to illustrate a tradeoff between two different issues.

4.1 Some Design Issues

The Hardware Environment

The project described in this report is the direct extension of a previous project. This version of the system has been developed and runs in a hardware environment designed to accommodate a variety of research projects in the computer graphics field. As such, little attention has been paid to the creation of an ergonomically well designed environment for this particular application. In the future, it is expected that a version of this editor will be available in a single-user workstation environment. During the search or development of such an environment, care must be taken to ensure user comfort. A number of studies have been done recently, particularly in Europe, that focus attention on the design of keyboards, video display terminals and workstations in general. A review of some of these issues is provided in [Rupp81]. A great deal can be learned from such work and applied to the design of graphic workstations as well. A choreologist using the editor described here would often spend hours entering and modifying Benesh scores and a physically comfortable environment would be important.

Interaction Devices

One aspect of the above design considerations is the choice and configuration of the interaction devices. These must be easy to use and appropriate for the task. Six types of interaction (select, position, orient, path, quantify and text) are itemised in [Foley84] and various input devices, such as a tablet, joystick, keyboard, lightpen, trackball and potentiometer, are examined in relation to each of these. The design of the editor described here makes extensive use of the select, position and path operations, and limited use of the text operation. A tablet with puck is employed for the first three and appears to work quite well. The Macintosh editor employs a mouse as the primary interaction device. Both systems use the keyboard for the input of text. Future enhancements to the editor are expected to include the orient operation as well (to arbitrarily rotate a Benesh sign, for instance).

Mode-Oriented Operation

A more fundamental difference between the Macintosh system and the editor described here is the degree of mode-oriented interaction. ChoreoScribe has been designed as a highly mode-oriented editor, primarily in order to accommodate the body-model aspect of the system.

When a sign is placed in the work-frame, the system must know which body part or which item of staging information is being specified. A natural method to specify this is to select an appropriate menu item. In this particular mode, the editor currently enforces certain constraints relating to the syntax and semantics of Benesh Movement Notation for the item in question. Such constraints include, for instance, the automatic crossover of a basic limb sign when it is not located in its own domain, or the placement of a sign in a valid position only. In the future, more rigid constraints can be added to the checking done by the editor. In this way, the system can relieve the choreologist of some of the burden of notation and help in the verification of what has been notated.

In some cases, a logical relation or similarity can be exploited to combine modes. For example, a leg may be defined by specification of the knee and the foot, by specification of the knee only, or by specification of the foot only. Rather than require the user to individually indicate both a knee and a foot, a single limb option is selected and, in this mode, either or both may be specified. The specification of both a direction faced sign and a turn sign have also been collapsed into a single mode due to the similarity in concept and interface design of the two operations.

For some operations in the editor a mode-less operation has been implemented. This appears to work well, for instance, in the modification option. Facilities must be provided for the modification of text strings, simple and multiple-symbol Benesh signs, in-frame and between-frame movement lines. Providing an individual mode for each of these tasks would have required five menu options and a more complex interface than necessary. A single Modify operation is provided instead. Upon selection of this option from the menu, the user may select the item in the work-frame to be modified. If a text string or Benesh sign is chosen, the cursor becomes that item and it may be repositioned in the work-frame. If an in-frame movement line is selected, the editor allows modification of that curve in the work-frame. Finally, if the portion of a between-frame locomotion line visible in the work-frame is picked, modification

of that curve is performed in the stave area of the display. Similarly, the Erase operation is a mode-less function used to delete any one of the above types of objects.

The general mode-oriented design of the editor is necessary to accommodate the body-model feature and, due to the extensive use of menus, reduces the load on the long-term memory of the user. It also provides a convenient, explicit scheme for the modularisation of the program functions. On the other hand, less mode-oriented operations place a slightly higher load on the user's long-term memory because the appropriate differences between the functions that share a mode must be remembered. Advantages of such functions include the simplification of the conceptual model by employing a common interface for similar operations on various objects (see 'Simplicity' section below).

Modularity and Layering

It is a commonly accepted fact that the use of structured design and programming techniques are helpful in the creation and maintenance of computer systems. The explicit modes described above provide a convenient scheme for the modular design of the program. Of course, in addition, the various functions of the program can be further sub-divided into sub-tasks that can, in turn, become modules themselves. A particular function can, for instance, be sub-divided into code that controls the user interface, code that manipulates the data structure, and code that provides the feedback (i.e. controls the display). Between the user interface code and the data structure and display code, it is useful to introduce a layer of abstraction specifying the logical information required. This organisation has led to a model found useful in the detailed design of the program which is similar to the ANSI/SPARC three schema architecture for database systems illustrated in Figure 4.1[Date81]. During the design phase of developing an interface for a particular function, one can determine what information is needed for data structure manipulation and display control. This information is obtained from intermediate variables. The user interface is then designed independently, to define the required data and store it in these intermediate variables.

Let us examine, for example, the case of the turn sign. During the design, it was decided that the data required by the display code and thus stored would be:

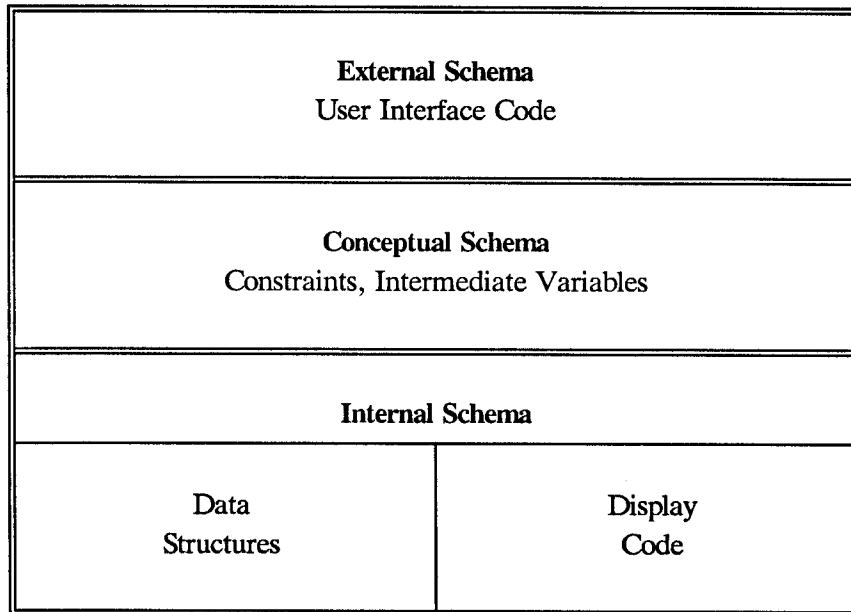


Figure 4.1 Layered Model for Code

- the amount (i.e. $1/8$, $1/4$, $3/8$, ... , $1-3/4$, $1-7/8$) and direction (clockwise or counter-clockwise) of the turn, and
- an angle of rotation of the final sign (because a turn can begin in any one of eight basic directions faced).

The intermediate variables from which this information is obtained is:

- a starting direction,
- the direction of tracker movement, and
- the amount of movement of the tracker.

Then, the user interface is designed to provide the data required in these intermediate variables. At this stage, input data can be filtered and constraints can be applied to ensure that only a valid sign is being specified. One can also, for instance, account for a change in the direction of the tracker movement. When sign specification is indicated as being complete, a separate section of code can calculate the amount of turn from the starting direction and the amount of tracker movement. The direction of the turn is obtained directly from the

direction of tracker movement. The orientation of the final sign is determined by the starting direction specified.

Not only does this scheme promote the modular design of the program, but it also has other advantages. The isolation of the code that controls the display of information makes it easier to modify this aspect of the program should some display primitives change. This may be the case if new software were to become available for the current environment or if the system is to be moved to another environment in which different display primitives are used. In other words, the program's portability is enhanced. Another advantage is to make the prototyping of the user interface easier since the code to effect this is independent of other code.

Prototyping of User Interface

For most functions of ChoreoScribe, the development of a user interface has been an iterative process. After the general concept of the user interface for a function is designed, it is implemented and then used by choreologists. In addition to discussions with the user, each is also observed while using the editor. These activities help isolate aspects of the interface that could be improved. Areas identified by observation are often not indicated explicitly by the user because some difficulties may be attributed to being unfamiliar with the system or to the facilities that new technology provides. In some cases, the improvements involve a 'fine-tuning' of one part of the interface, such as the reshaping or re-ordering of a menu, or changing the placement of a menu. In other instances, however, alternative approaches to the specification of the required information result and a new approach is designed, implemented and again tested. This process is summarised by others, such as Foley, as follows: the semantics of the interface remains constant but the syntax may change [Foley82a]. Brooks goes so far as to say: when creating a system, "plan to throw one away; you will, anyhow" [Brooks75]. Throughout the prototyping phases of all functions of the editor, the choreologists involved in the project have had a controlling consultive role in suggesting the design of the user interface and approving the final implementation.

Simplicity and Consistency

When reviewing human factors literature relating to computer systems, simplicity and consistency are often described as being fundamental to the acceptance of the system by the user. These factors are indeed important. Both simplicity and consistency are achieved by adhering to a few specific principles during the design of the various functions of the system.

One way to make a system appear simple is by appropriate definition of the *conceptual model* with which the user must deal [Singh82a, Foley84]. This is the set of ideas which are used by the user to explain the behaviour of the system. Study of the way in which choreologists currently work can lead to the use of similar mechanisms in the computerised system. Thus, employing concepts already familiar to the user will make the system easier to learn. This approach, however, can be a double edged sword by possibly trapping the user (and designer) into functional fixedness. In some instances, computerisation of a system can facilitate improvements in current methodologies employed by the user. However, the use of these new, more efficient methods may meet some resistance on the part of the user. They require that the user's conceptual model adapt itself to the new system. Care must be taken when introducing a method to which the user cannot relate prior knowledge or experience. In many respects ChoreoScribe has been designed to take the choreologist's conceptual model of Benesh Movement Notation into account.

One of the basic design principles of ChoreoScribe is 'pick-and-place.' In the current manual procedure of creating a score, the choreologist draws each Benesh sign. Using ChoreoScribe, the user is presented with a menu of available signs and simply picks the desired sign. The cursor then becomes that sign and may be dragged into the work-frame and placed in the appropriate position. This is one approach that does differ from the conceptual model of the current manual procedure. This 'pick-and-place' type of operation is possible in many cases because there are a finite number of options available to the user.

On the other hand, a free-hand, in-frame gesture line may take an infinite number of shapes. The operation here is similar to the manual procedure used to specify such a line. After specifying the appropriate option, the user sketches the desired line much as would be done with pencil and paper. This was considered the most natural approach to the creation of such lines. Here, the design of a simpler, more natural interface resulted in a more complex

implementation of the function. It is often the case that software that is easy to use is difficult to create [Edwards83].

If the notation does not predefine a position within the work-frame for a particular type of sign, the user must drag it into the work-frame and place it appropriately. If, however, the notation does define the position of a sign, that sign is placed automatically by the system in order to simplify the operation. These two concepts correspond to Foley's *closed loop* and *open loop* tasks [Foley84].

Another way to achieve simplicity is through *consistency*. One way to achieve consistency is to ensure that a particular operation is used in the same manner throughout the system. This allows the user to apply the knowledge and experience gained about one aspect of the system to other aspects. One example of an operation that has been implemented consistently in ChoreoScribe is the 'pick' operation. To select an item (a stave line, a frame, a Benesh sign, a curve or a menu item), the user aligns the tracker symbol (an arrow) with the desired object and presses the yellow *do* button on the puck. Another example is the Modify operation. As discussed earlier, to move a sign, the user selects the desired sign by pointing to it with the tracker and pressing the 'do' button. The tracker then becomes the indicated sign. It may then be dragged to another position and placed by again pressing the 'do' button. In an entirely different context, if the user wishes to modify a curve by moving a control vertex, that vertex is again picked, dragged and placed as described above. One exception to this rule is the moving of frames in a score. It would be infeasible to have the tracker become the selected frames to be dragged to their new position. Instead, after the frames are selected, the new position is chosen with the arrow tracker and the frames automatically inserted. In this tradeoff between consistency and simplicity, the latter was judged to be more important.

In another tradeoff situation, consistency was deemed to take precedence over simplicity. This case involves the drawing of free-hand sketched gesture lines. One approach would have the user press the 'do' button at the beginning of the line, hold it down while sketching and release it at the end of the line. This may seem to be a more natural analogy to the current manual method of drawing a line and is the principle employed in the Macintosh version of the Benesh editor (for sketching curves and dragging signs). However, to remain consistent with the operation of all other functions of ChoreoScribe, the user presses and releases the 'do' button at the beginning of the line, sketches the path

of the curve and again presses and releases the 'do' button at the end of the line.

The following are further examples illustrating consistency throughout the system. The first of these is the use of puck buttons. Throughout the program, the top (yellow) button is the *do* button as described earlier. The right (green) button allows the user to abort the current function, and pressing the left (white) button results in a 'help' menu appearing on the auxiliary terminal screen. Another example is the availability of only those menu options that are valid in the current state of the system. There is one exception to this rule discussed later. Error messages also always appear on the terminal screen and are accompanied by a beep.

Finally, adherence to the principles of simplicity and consistency is also important during the design and implementation of the software itself. Writing code that is more efficient but less readable adversely affects the maintainability of the system. Simpler, more readable code can be more easily understood both by the original programmer and others should modifications or extensions be required at a later date. Similarly, adhering consistently to a set of programming conventions also increases the maintainability of the program.

Error Correction

The previous prototype of the editor included a facility for quick error correction in the form of an *undo* function. This was activated by pressing the bottom (blue) button on the puck. This 'short-cut' is recognised as necessary in a full working version. However, due to time constraints and the desire to explore other aspects of the human interface, it has not been maintained in this version of ChoreoScribe. The user must use the available facilities to undo any erroneous operation.

Error Avoidance

Another fundamental principle of ChoreoScribe is that of error avoidance. This is a highly desirable objective in that it minimises the amount of time spent by a user in error recovery activities. This property must be considered during the design phase of any interactive system. In our case, a number of methods have been employed to prevent errors.

The most obvious of these is to display only those menu options that are permissible at a particular point in time. For instance, it does not make any sense to Delete, Move or Copy frames if the score is empty. These options are not available if this is the case. The Next Line (scroll) operation is only presented if another score line exists after the lines currently displayed. Similarly, the Previous Line option is only visible if a score line is present prior to those displayed. During the modification of a movement line, one of two menus is displayed depending on whether or not the control vertices (CV) are visible. If they are not, a menu with the options (i) Done, and (ii) Show CV is shown. Otherwise, a menu containing (i) Done, (ii) Hide CV, (iii) Add CV, and (iv) Remove CV is shown. A control vertex may also be picked and moved at this time. Clearly, one does not want the option to Show CV if they are already visible and vice versa. One exception to this rule is found in the case of the stage location menu. The menu consists of nine symbols representing various columns of the stage area. If the performer is located in one of these columns, the appropriate symbol is picked. If, however, the performer is located between two columns, the resulting sign contains the corresponding two adjacent stage column symbols. In order to remain consistent with the above rule, after selection of the first symbol, only the two adjacent symbols should be shown as valid choices. Instead, the entire menu is displayed again for the second selection. This was done here to more clearly illustrate the context of the operation being performed. This is thought to be more helpful than strictly adhering to the above error avoidance technique. Feedback indicating the first selection is provided.

Other examples of error avoidance include the automatic placement of Benesh signs when the notation predefines their position. This helps prevent syntax errors in the score. Another example is the automatic crossover of basic limb signs when they are located outside of their own domains.

Future work will also address the Error Avoidance issue. Further constraints could be applied to enforce the proper syntax and semantics of Benesh Movement Notation. This will become feasible when a full body model has been implemented.

Minimising Motor Movement

As outlined earlier, minimising the amount of effort for motor movement is one of the objectives of a good user interface. This issue has been addressed through a number of design decisions made. Throughout the design of ChoreoScribe, minimising puck movement and the number of button pushes required to accomplish a task has been a major objective. Wherever feasible, sub-tasks of a function have been automated that would have otherwise required active involvement of the user. A case in point is the automatic placement of signs for which the notation predefines the position in the frame. This reduces both puck movement and button pushes.

Another example illustrating attention to this concern is the placement of menus. In general, the placement of pop-up menus is dictated by the current position of the tracker. These menus usually appear as a result of the selection of a static menu option. One could have centred the pop-up menu directly on the cursor position, but this would have resulted in the superimposition of the pop-up menu on the option of the static menu. For clarity, it was decided to display the pop-up menu immediately above or below the cursor position in most cases. A small amount of puck movement then positions the tracker over the options of the pop-up menu. This placement of menus also minimises the movement of the centre of attention about the display screen.

The specification of four different body parts, the limbs (arms and legs), each use a similar pop-up menu. Because of the similarity of these menus, it was decided to consistently place them at one position on the screen (near the body menu). This pop-up menu allows specification of a bend and an extremity. After picking a sign to define either of these two items, the sign is dragged into the work-frame and placed. At this point, the user may specify the other item of the pair or return to the static menu to continue with another function. In order to continue, a 'done' option has been provided as part of the pop-up menu to indicate completion of limb specification. Initially, it was placed near the other options of the menu in order to provide a mental association with the menu. After placing the first sign, the tracker was moved from the work-frame back to the menu to select the 'done' option. This required substantial puck movement. Instead, the 'done' option now appears to the side of the work-frame near the position of the sign just placed. If further specification of the limb is not required, only a small motion puts the tracker over the 'done' option.

Learning Aids

It is expected that ChoreoScribe will be used primarily by persons with limited computer experience. As such, efforts have been directed towards exploring some ways of assisting the user in learning and using the system. One way to do this is provide the user with feedback about the system state and activity. This is done to prevent user frustration or panic. This issue is discussed in more detail later.

A system of help menus has also been implemented in this version of the program. The various help screens may be found in Appendix C. The help menus are accessed by pressing the white puck button. If the tracker is situated over a menu option of a static menu when the white button is pressed, a brief description of that function appears on the auxiliary screen. If the user is in the midst of a function and presses the white button of the puck, the editor again briefly describes the current function or sub-task of the function. Finally, if neither of the above are true when the white button is pressed, a one line description is displayed for each of the functions available in the current program state.

4.2 A Menu Driven System

Interactive computer systems can often be divided into two groups: those with a user-initiated dialogue and those with a computer-initiated dialogue. User-initiated dialogues include mnemonic based and natural language based systems. These are often employed for systems in which the user community consists of computer trained and/or experienced persons. The user requests an action by issuing commands to the system. Computer-initiated dialogue systems, such as menu type and question-and-answer type systems are more appropriate for an untrained, novice user community. Such systems are easier to learn and use by less computer-experienced users. Here the user is prompted for action by the system, which, in effect, guides the user.

Memory Load

The use of any computer system requires the memorisation of information. Load on the user's memory comes in two forms: short-term and long-term. Short term memory is employed by the user to retain unprompted knowledge of task elements while performing a function. Long-term memory is required to recall the details of a task. In other words, if the task being performed involves a number of steps which the user must remember, long-term memory is employed. The above two types of interaction dialogue place different loads on the two levels of memory.

Since the capacity of short-term memory is limited [Miller56], and the use of a menu driven system reduces the load on short-term memory, the menu driven approach has been taken in ChoreoScribe. Depending on the system state, relevant menu options and other information are displayed to act as a 'visual cache'. This system-initiated type of interaction lends itself well to incorporating effective feedback while a task is being performed. The user is guided through the steps of the task by the system, and at the completion of each step feedback is employed to keep the user informed of progress toward task completion. Menu driven systems are acknowledged to be easier to learn and use than command driven systems, especially for less experienced computer system users. One disadvantage of a strictly menu driven system is that the sequence of steps required to complete a task is often fixed and the user is not free to find possible

short-cuts.

In user-initiated dialogue systems, users are free to accomplish a given task in any way they wish using the available tools. Thus, the user may use methods that are found to be particularly efficient. However, the user's long-term memory is taxed with remembering what tools are available and how they are used. The flexibility offered by such a system may best be taken advantage of by regular and experienced users of the system. A system-initiated dialogue was judged to be more appropriate in this application.

ChoreoScribe is primarily a menu driven system. The majority of tasks (or functions) implemented in the editor consist of a number of sub-tasks. Upon selection of a function, the user is guided through the appropriate sequence of steps to accomplish the indicated task. A few tasks, however, are very simple and are more appropriately implemented using a user-initiated dialogue. These are the request to abort a current function and a request for the display of help information. These two operations only require a single button push and may be invoked at any time.

Static and Pop-up Menus

Static menus are those that are always displayed at a fixed location on the screen. The user learns to expect such a menu to appear in its fixed location. This has the advantage of "muscle memory" [Foley84], in which the user's attention automatically shifts to the position of the menu when it is displayed. In fact, experienced users of a system often move the tracker to the position of a particular menu item *before* it is visible. Two disadvantages of static menus in ChoreoScribe are that (i) the user's visual attention must shift to the new screen position, and (ii) they occupy display space, which is at a premium. In ChoreoScribe, making some static menu items invisible and highlighting certain others is used to provide feedback to the user regarding the system state.

A pop-up menu (or dynamic menu or moving menu) always appears near the position of the tracker on the screen. In this way visual continuity is preserved. Both hand and eye movements to the menu are minimised. Since pop-up menus may appear anywhere on the display screen, they may overlay existing information on the screen. Thus, they often occlude other data (as is the case with the Macintosh editor), or the menu options may be highlighted to make them distinguishable from other information displayed on the screen.

ChoreoScribe employs both types of menus. Static menus are used in the first two levels of the menu hierarchy and pop-up menus are used at the leaves of the hierarchy.

Hierarchy of Menus

Due to the number of functions currently available in ChoreoScribe and the expected expansion of its functionality, it is not feasible to use a single-level menu structure. Instead, a hierarchy of menus has been implemented. The first two levels of the hierarchy consist of static menus described above (the stave-menu, the frame-menu and the body-menu). When the third level has been reached, an individual Benesh function has been selected and these are implemented using pop-up menus. An individual function may consist of a single pop-up menu or its own hierarchy of pop-up menus. Studies have shown that productivity is higher when broader menus with fewer levels are used [Snowberry83]. Many Benesh functions in ChoreoScribe are completed using three levels of menus. In the future, the functional extension of the editor may involve the use of more menu levels. Although the hierarchical menu structure should be maintained to assist in the learning process, it may become feasible to provide experienced users with some form of more direct access to the desired functions.

Text and Icons

ChoreoScribe makes extensive use of menus, each consisting of a number of options which may be represented either by a text string or a graphical figure. The graphical representation of a command or operand is known as an *icon*. In recent developments in the computer industry, a great deal of emphasis has been placed on the use of icons (e.g. the Xerox Star, Apple Macintosh and Lisa computers). The very nature of Benesh Movement Notation is to provide a visual representation of information. The various Benesh signs provide a convenient means to represent many menu options. These signs are employed in two ways. First, a sign may be used to explicitly represent a menu option (an operand). To specify that a performer faces upstage, the upstage sign is chosen from a menu (Figure 2.6), for example. Secondly, in some cases, the semantics of a sign (as understood by the choreologist) may be used in a sub-task of defining a different sign. An example illustrating this is the specification of a turn sign. To define a turn sign, both the beginning direction faced and the ending direction faced must

be specified. To perform these two steps, the menu of direction faced signs is used. When selecting a sign from the menu in this context, one is defining the starting or ending information required for the turn sign rather than specifying a direction faced sign.

The use of icons, however, does not always decrease the cognitive load of menu selection. The advantage of an icon is realised only if it more effectively expresses its meaning than the corresponding text string. Some items of information are better expressed in textual form. A number of pop-up menus in ChoreoScribe include a 'done' option. This option is presented in its text form even though all other options of the menu are represented using an icon. The options of the stave-menu and the frame-menu also use text strings to label the operations or functions available (see Figure 3.3). Key words indicating operations, such as 'Archive', 'Edit Frame', 'Delete', 'Save', etc. are used in the stave-menu. Benesh terminology, or an abbreviation thereof, is used to label options of the frame-menu such as 'Location', 'Dir. & Turn' for direction and turn data, 'Mus. Scoring' to specify musical scoring information, etc. The body-menu (Figure 3.3), on the other hand, is an excellent example in which the pictorial representation is better than using the text equivalents. In contrast, the Macintosh version of the Benesh editor uses icons predominantly.

Number of Menu Items

One method by which the number of levels in a menu hierarchy can be reduced is by including a greater number of options at each level. When determining the number of options to appear in a menu, a rule of thumb often used is: "Seven plus or minus two" [Miller56]. Throughout the design of the human interface of ChoreoScribe, this principle has been adhered to as closely as possible. In some instances, in order to reduce the depth of the menu hierarchy, this rule has been stretched by placing more than nine menu items into a number of logical clusters. A case in point is the musical scoring menu. Depending on the system state, this pop-up menu may contain up to 19 options. These options are grouped as follows: nine possible signs (digits) to specify a time signature, six possible rhythm indicators, three types of bar lines, and one option to specify that a repeat sign is to be defined. Each of the four clusters now contains no more than nine options. By grouping the menu items in this logical manner, the user can quickly recognise the cluster of interest and find the desired option within that cluster. The reader will note that the stave-menu and frame-menu each

contain 13 options (see Figure 3.3) that have been grouped logically into sub-menus, again consisting of no more than nine menu items.

Menu Shape and Ordering

There are a number of ways to list menu items: both shape and ordering are parameters. Menu items may be placed in a horizontal list, a vertical list, a circular list, blocked or clustered into groups, or in some random shape. Studies have indicated no clear advantage between horizontal and vertical lists [Coffee61, Earl65]. In *ChoreoScribe*, all menus composed of textual options are organised in a vertical fashion. These include the static stage-menu and frame-menu, and the pop-up menus for movement lines. Clustering, with extra space between the groups, has also been employed because this has been shown to improve search time for a menu item [Cropper68]. The previous version of the editor employed a circular list for all pop-up menus regardless of content. In the current version, the content of the menu and the logical ordering of the menu options (discussed below) determine the shape of a pop-up menu.

Within a menu, the items may be ordered lexically (text only), by frequency of use, randomly, or in some logical manner (related options placed together). The ordering of textual menus in the editor is done in a logical manner wherever a logical pattern could be devised. For pop-up menus the order and shape are determined by the information contained in the menu. Some examples to illustrate this are described below. The stage location menu contains, from left to right, nine signs indicating stage columns from stage left to stage right. The direction faced menu is a circular menu in which, when superimposed on a diagram of the stage, each sign illustrates the direction faced represented by that sign. A sub-menu of the travel sign is organised in the same manner to illustrate travel in any one of the eight basic directions. The menu of digits used to specify time signature or number of repeats is organised in a fashion similar to that found on touch-tone telephones.

4.3 Providing Feedback

In any form of conversation, be it between two persons or between man and machine, feedback plays an essential role. Providing the user of a system with appropriate feedback is an important part of the human interface of any interactive program. Feedback, used effectively, can help reduce user boredom, confusion, frustration and panic [Foley82b]. In ChoreoScribe, feedback is used to keep the user informed of the system state, the current status of the score, and the current status of the work-frame. Foley has developed a model which specifies three levels of feedback to the actions of the user:

- lexical,
- syntactic, and
- semantic [Foley82a].

Lexical feedback is the lowest form of feedback. Any lexical action or input performed by the user can be provided with a corresponding lexical output. Examples of this type of feedback include: echoing of characters on the terminal as they are entered, moving the tracker in response to movements of the puck, and illustrating the path the tracker has followed when a curve is sketched.

Feedback at the *syntactic level* is a response to a completed input action by the user indicating that the system has accepted the input. When an option is selected from a menu and that option is highlighted, the user is informed that the action has been accepted and understood by the system. Other forms of syntactic feedback found in the editor include: indicating the position in the score of the current work-frame, prompting for the next input (e.g. a pop-up menu appears), and changing the tracker to a sign that has just been chosen from a menu.

Finally, *semantic feedback* informs the user that the requested task has been completed. This is usually done by displaying the results of the operation just performed. In our case, this is accomplished by displaying the work-frame after a sign has been placed or a curve has been defined, and by redisplaying the score lines after a frame has been inserted or modified. Another type of semantic feedback is employed if the requested task will take more than a few seconds. This is the case for such operations as archiving and de-archiving a score. Here, a placebo is used to tell the user that the system is working on the requested task.

The placebo used in ChoreoScribe is a Buddha icon[Singh82a], adopted from Newswhole [Tilbrook76]. In addition, for every line of the score read from or stored to the disk, another form of feedback is provided (a dot appears on the auxiliary screen). As the operation is underway, a string of dots grows along the bottom line of the screen. In this way, the user can watch the progress of the operation. In a similar fashion, when the editor displays a line of the score, rather than create the line internally and then display it in its entirety when it has been completed, it was decided to show the line as it is being formatted and the display created. In this way the user can see that the system is actively working on the task of displaying the line.

Let us now examine one single function in detail in the light of these three levels of feedback. The turn sign is defined by specifying (i) the beginning direction faced, (ii) the direction of the turn (clockwise or counter-clockwise), (iii) the ending direction faced, and sometimes (iv) the number of turns. The steps used to define this information are:

- selecting the beginning direction faced from the direction menu,
- moving the tracker (and thus the puck) in a clockwise or counter-clockwise direction,
- selecting the ending direction faced from the direction menu, possibly
- selecting one of four turn signs complete with number of turns indicator, and finally,
- dragging the completed sign into the work-frame and placing it.

Lexical feedback is provided in the second step by having the tracker follow the movements of the puck and by actually drawing the path that the tracker follows. In the fifth step, lexical feedback is again provided by having the completed sign follow the path of the puck (dragging the sign). Syntactic feedback is provided by displaying the appropriate menus at steps one, three and four. The tracker taking the form of the completed sign in the fifth step is also a form of syntactic feedback. Finally, semantic feedback is provided by displaying the completed sign at the chosen location in the work-frame.

As outlined above, feedback is used primarily to provide three types of information, which we will examine in turn; (i) the current system state, (ii) the current status of the score, and (iii) the current status of the work-frame. Providing feedback of the *system state* takes two forms. Selecting an option from

a static menu causes that menu item to be highlighted (inverted) and all other items in the menu to be switched off. This has the effect of illustrating the path taken by the user through the first two levels of the menu hierarchy. These are used to indicate the mode the user is currently in (e.g. Edit Frame + Location, Edit Frame + Left Arm, Add Frame + Left Arm, etc.). Another method used to indicate the system state is altering the shape of the tracker icon.



select a command



select an object

If the tracker icon is a cross-hair, a command is to be selected. If the tracker icon is an arrow, an object (frame, sign, position, etc.) is to be selected. Finally, if the tracker icon is a Benesh sign, that sign is to be placed in the work-frame.

Feedback is also used to show the *current status of the score*. Some examples of such feedback include the following. The editor displays the two score lines being edited in the display window at all times. The frame being modified or the insert position of the current work-frame is indicated with an arrow in the display window. The position of the two displayed stave lines within the entire score is indicated by a scroll bar appearing to the right of the stave lines. Each frame of the two displayed stave lines is shown in its current state (with the exception of the frame being edited; it is updated in the stave lines when editing of the work-frame is finished). Between-frame locomotion movement lines that have not been completely specified are indicated with a short dashed line.

The *current status of the work-frame* is also indicated through the use of feedback. As discussed earlier, the editor has two modes of operation: a full body model mode and a notation mode. In the former, all body parts that have been defined are shown in the work-frame. This includes those that have been carried from a previous frame and those that have been newly specified in the current work-frame. In the notation mode, generally only those signs that are different in the work-frame from those found in the preceding frame are displayed. In this case, signs carried from previous frames are defined but not

displayed in the current frame. A menu option has been provided to allow the user to switch between these two modes of operation. To distinguish between items that have not been defined in the current frame, those that have been carried from a previous frame, and those that have been newly defined, different colours are used to highlight the corresponding menu option. If an item has not been defined, the background of the menu option is left black. If an item has been carried from a previous frame, the background colour of that menu option is grey. Finally, when an item has been specified in the current work-frame, the corresponding menu option is highlighted with the background colour of the work-frame.

One more point should be made before we leave the topic of providing feedback. As indicated above, feedback should be provided immediately. For operations that are expected to take more than one or two seconds, some form of placebo should be provided. The designer of the program can also take response time into account to some extent. Maintaining information for the display of a score line is a case in point. Data widths must be maintained for each frame of a line. These could be calculated each time a score line is to be displayed. This would require that a fair amount of computing time be added to an already time consuming task. Instead, the data width of a frame is recalculated each time the display contents of that frame changes. In other words, whenever a frame is inserted or modified into the score, the data widths of the affected frames (the frame itself and its successor) are re-evaluated. In effect, the computing load is distributed into shorter delays that do not noticeably affect the response time so that a longer delay can be avoided later.

Colour Versus Resolution

Although current development of the editor has taken place using a powerful, medium resolution (512 X 512) frame buffer, the necessity for a higher resolution has become evident. Guidelines for the production of Benesh signs specify stroke widths between 0.18 mm and 0.35 mm [Grater84]. Composite signs displayed on the current system lose some detail due to truncation error when mathematical rotations are applied. A higher resolution display would also prevent *sign collision* when the score is displayed. The work-frame in which the details of a frame are specified, is four times the size of the frame in the actual score. It is possible to place two signs next to one another in the work-frame with a discernible space between them (e.g. a single pixel). When that frame is

displayed at one quarter the size in the score lines, these two signs may collide. Although the printing of these on a high resolution output device again shows the space between the signs, it would help prevent user frustration and panic if signs and curves could be displayed more accurately. It appears as though only a monochrome display system would satisfy the two requirements of obtaining an appropriately high resolution (1024 X 1024) and keeping the projected cost of a workstation environment within reason. As well as being aesthetically pleasing, the use of colour is helpful in this implementation of the editor, as discussed above. The tradeoff between high resolution and availability of colour has been discussed with some choreologists. The general consensus favours increased resolution over the use of colour.

4.4 Editing Curves

Curved lines are used in ChoreoScribe to represent various forms of movement. These lines may be contained entirely in a single frame or may link two frames of a score. For a detailed description of the rendering of these movement lines, the reader is referred to the previous chapter. The interactive manipulation of such lines as discussed in [Bartels84] has influenced the development of the human interface here.

Some movement lines are sketched by the user as they would be drawn with a pencil and paper. In this case, the user moves the tracker about the work-frame, and feedback is provided by showing the path followed by the tracker with a series of dots. The interval of sampling was originally time-based. As the user moved more slowly for complex sections of the curve, more points would be collected. Conversely, rapid movements over simpler sections of the curve resulted in fewer points being collected. This mechanism often resulted in an unnecessarily large number of points being collected, increasing the complexity of later computations. If the user stopped for a few seconds during the sketching of a curve, a number of sample points in the same vicinity would be collected. Because of these problems, time sampling was abandoned. Distance-based sampling is used in the current version of the system. In this case, the collected points along the path are separated by a relative uniform distance. A least squares fit is obtained to the collected sample points and the resulting curve is represented using B-splines defined by a sequence of control vertices.

Other movement lines (such as steps, jumps and slides) take on a general shape defined by the notation. In these cases, the editor makes an initial guess for the shape of the required line. These have come to be known as “canned” lines. After some experimentation, basic rules were formulated to specify the position of the control vertices for each of these types of lines.

In both cases, we have a curve represented by a B-spline defined by a sequence of control vertices. These do not, in general, lie on the curve itself but form a control polygon that contains the curve. The user must be able to modify the shape of a movement line. This can be done in a number of ways: increase or decrease the number of control vertices used to generate the curve or move one or more of the control vertices. These functions have been provided in

ChoreoScribe. If the user indicates that a line is to be modified, a menu containing options representing the above three operations appears.

Free-hand drawn curves are first represented by four control vertices in order to smooth out the sketch curve as much as possible. The sketched curve may contain intended complexities such as a loop, for instance, which will not be accurately represented by the resulting curve. More control vertices (see below) are needed to more accurately approximate the drawn curve. Canned curves (steps and jumps) are initially generated with five control vertices. Moving a control vertex affects the neighbourhood of the curve near that control vertex. This is done by selecting the appropriate control vertex with the tracker and dragging it to its new position. The curve is not redrawn in real time as the vertex is moved as this is computationally too expensive an operation. The new curve is drawn only after the vertex has been repositioned. Increasing the number of control vertices makes the neighbourhood controlled by each vertex smaller. Conversely, decreasing the number of control vertices causes the area of the curve affected by each vertex to increase. These two operations are available through the selection of menu items.

The manipulation of curves in this way becomes quite natural with some experience. Choreologists experimenting with the system have grown accustomed to this mechanism. Perhaps further research will yield the mathematics by which the user may drag part of the curve itself (rather than the related control vertex) to affect the desired change.

Finally, in-frame movement lines can be modified in the work-frame but only a small portion of a between-frame movement line is contained in any one frame. If such a line is to be modified, it is selected in the work-frame. If that curve appears in one of the stave lines, it is highlighted in the stave line and modification takes place there. This is done in order to provide context for the movement line being modified. A movement line does not appear in the stave lines until the frame in which it ends is inserted (or updated). In the case of a movement line which does not yet appear in the stave lines, its shape is modified in a work-area at the right of the screen (Figure 3.3).

5 Future Extensions

The work described here is but a small part of a system that may eventually find acceptance as a useful tool for the notation of dance using Benesh Movement Notation. The current prototype has already illustrated its usefulness in the error correction of scores as they are entered. Further work on this project will make the application of such an editor as a tool to create scores more feasible and enhance the process of editing a Benesh score. This chapter discusses extensions to this project in four major areas:

- the implementation of the editor in a workstation environment,
- an extension of the full body model,
- the expansion of notational functionality, and
- the enhancement of editor functionality.

5.1 A Workstation Environment

The primary objective of this editor is to provide a tool to be used for the notation of dance works. Most choreologists work in conjunction with a dance company. As a result, the editor must be available to both individual choreologists and to dance companies. At its current stage of development, the editor runs on a much more powerful and expensive computer system than is necessary. The environment is also a timesharing one. To be feasible as a tool for choreologists, the editor must be available on an inexpensive, stand-alone workstation that guarantees real-time response. The current editor does not use the frame buffer to its fullest capabilities in the hopes that moving it to a workstation environment would be easier later. There are a number of requirements that should be satisfied by a workstation environment. If that environment is to be used for further development of the editor as well as productive use, additional requirements should be met. We shall discuss some of the issues pertaining to:

- an applications processor,
- main memory,
- external storage medium,
- a frame buffer and monitor,
- an input device such as a tablet with puck or a mouse as well as a keyboard,
- a high-quality output device,
- an operating system, and
- software tools to support development and maintenance of the editor.

An appropriate workstation environment may be available as a complete unit from a single vendor. Alternatively, the required workstation can be built using components supplied by a number of vendors.

The *applications processor* should have at least a 16-bit wide data path. The majority of processors used in workstations today have either a 16-bit or 32-bit word-size. The large address space provided by such processors is needed to

allow sufficient memory on the system. As a dance score increases to its full length, the data structures used to store signs, curves, frames and score lines require a large amount of data space. As computer memory is relatively cheap, and decreasing in cost, it is more effective to provide programmers with a large amount of memory rather than forcing them to deal with memory shortage problems.

If the system is to be built using components from various vendors or extension to a given workstation is anticipated, care must be taken in the choice of *internal bus* used. Intel's Multibus has become a popular choice and is currently supported by numerous vendors. It does have one drawback in that it employs a 16-bit wide data-path. With the increasing popularity of 32-bit processors, some vendors are turning to the use of larger buses.

It is expected that between 500 kilobytes and one megabyte of *main memory* above and beyond that required by the system would be sufficient for the editor.

In addition, two types of *external storage* are required. First, a reasonable amount of disk storage is required to store scores being edited. A hard disk would provide fast, quiet access to scores, system utilities, etc. A disk, such as the currently available Winchester type, that provides ten or twenty-five megabytes of storage would be required. A capacity of ten megabytes could store up to 1500 stave lines. Secondly, the ability to transfer scores from one system to another is required. One alternative would be to include networking capabilities. This seems inappropriate in this application. A second, more appropriate, alternative would be to use floppy disks. Scores could be stored on such disks, copied and physically moved from one system to another. Currently available systems include, for instance, a 5½ inch floppy disk with a capacity of 0.3 megabytes and an eight inch disk with a one megabyte capacity. It is expected that a system with at least one megabyte of storage per disk (approximately 150 score lines) would be most appropriate. In addition, this floppy disk system could also be used for backup purposes.

The *frame buffer* and associated display processor should be capable of displaying a high resolution image. Although current development of the editor has taken place using a powerful, medium resolution (512 X 512) frame buffer, the necessity for a higher resolution has become evident. In order to obtain an appropriately high resolution (1024 X 1024) and keep the projected cost of the workstation within reason, it currently appears as though only a monochrome

display system would satisfy both requirements. It is expected that, as hardware costs decrease, a high resolution, colour display system will become feasible for this application in the not-too-distant future.

Another factor to be considered here is the physical size of the monitor. Choreologists would feel most comfortable if stave lines were displayed in their entirety and at their actual size as found on standard preprinted choreology notation paper. As a result, a monitor with a viewing surface (screen) at least 12 inches wide is required. Many monitors available today are either 14 inches or 19 inches wide.

Currently, the editor uses a tablet and puck as the primary *input device*. This seems quite appropriate for this application since it is a convenient and natural form of interaction for choreologists. For example, the interface design to draw curves corresponds closely to drawing such a curve using a pen on a piece of paper as the choreologist currently performs this task. A current popular alternative might be to employ a mouse. This is done in the Macintosh project. Although technologies, such as touch-sensitive screens or speech recognition, may become feasible for this application in the future, the use of tablet and puck or the mouse appear to be quite adequate. In addition, a standard keyboard is necessary to input certain types of information such as score titles and textual data. Obviously, if the workstation environment is to be used for editor development, enhancement and maintenance, a keyboard is required.

Eventually, a high quality *output device* will be required to make hardcopy output of the scores created. A resolution of approximately 200 dots per inch is seen as the minimum for the production of publishable quality scores. Devices with the required graphics capabilities and resolution currently exist but are relatively expensive (\$5,000 to \$7,000). It is hoped that further developments in the laser-printer field will result in lower costs for such devices in the future.

A number of factors influence the choice of *operating system* for the workstation. First, it must provide real-time response for activities performed while using the editor. This is a difficult criterion to quantify. One measurement of computer speed often employed is the number of (million) instructions per second (MIPS) performed by the hardware. However a great deal of emphasis is now being placed on research into the many factors that influence user productivity [Foley82b, Shneiderman84, Tombaugh85]. This may lead to a more appropriate measurement of the performance of a given system.

What is important here is that the operating system is designed to provide real-time response to actions performed by the user.

In addition, if the workstation environment is to support further development of the editor, a popular operating system such as UNIX has the advantage of a large number of existing software engineering tools such as editors, source code checkers, etc. being readily available. BCPL-based languages such as C have proven themselves reasonable as development languages for projects such as this and are easily obtained for such an operating system. An alternative approach, and that employed in the Macintosh project, is to do system development in one environment and execute the program in another. For example, although an editor has been designed to run on the Apple Macintosh personal computer, its development is being done on a VAX system running the UNIX operating system. In this case the program is coded and cross-compiled on the VAX and then transferred to the Macintosh for execution.

Another promising alternative is the use of the Harmony operating system [Gentleman83, Gentleman85]. This operating system supports a multiple process structure that could be useful in the implementation of this editor. Current research at the University of Waterloo, including [Forsey85] is exploring the use of Harmony. It has been implemented on a number of processors including the Motorola 68000, 68020 and the National Semiconductor 32016. This may prove promising as an operating system environment for the editor.

Finally, the cost of the system is an important factor. It is hoped that, initially, the required workstation environment may be provided at a cost of approximately \$15,000. This appears possible at the current time. In the future, it is hoped that the hardcopy output device can be included in this price. Complete workstations for this application are available from numerous vendors such as Masscomp, Orca, Sun, Tektronics and Wyse Technology. In this case it may be difficult to find a configuration that provides or supports all the facilities previously outlined. A second alternative would be to purchase the required components from various vendors and build a custom workstation to meet the required specifications. For instance, the Multibus architecture is supported by numerous vendors supplying processors (e.g. National Semiconductor, Motorola), memory (e.g. Chrislin Industries), external storage devices and frame buffers (e.g. Matrox, Ramtek). This approach has been taken in a number of projects in the past [Fisher84, Yamada84]. In the near future, a workstation to be used for psychology experiments is to be developed in the Computer Graphics Lab at

the University of Waterloo. Knowledge and experience gained in that project may perhaps prove useful to the Benesh project.

5.2 The Body Model

Let us now turn our attention to the editor itself. As previously discussed, although the system's primary function is to produce notation, it also maintains enough data to support a full body model. The implementation of a body model would have two immediate applications. The first is the *verification* of body postures and movement. There are two aspects to such verification: (i) whether what has been specified is notationally correct, and (ii) whether it is bio-mechanically possible. If the notation input is not correct or possible, the user may be informed of the error and correction can be effected immediately. Some research has been done [Singh82a], some is underway, and more is still required in the specification of the body model as it relates to Benesh Movement Notation. Certain aspects that must be accommodated by the body model include the differing size, body proportions, flexibility, strength, etc. of two performers executing the same score of notation. Some body model research, independent of Benesh Movement Notation, has been described by [Badler80, Calvert82, Herbison-Evans79].

A second application of the body model is in *animation*. Frames in a dance score define what is known to film animators as a *key-frame*. Animation systems employ some form of interpolation to calculate a number of intermediate positions between key-frames. Some such techniques for generating smooth animated motion are discussed in [Kochanek82]. Such a system could be applied to a Benesh score created by the editor to produce an animation of the dance using, for instance, a stick figure. It is easy to envision that such a facility could, for instance, be used in the creative process of choreographing a dance. It could also lead to the use of Benesh Movement Notation for other applications such as film. If such a system can control an animated two dimensional figure on a screen, could it also, for instance, control the motions of a robot in three dimensions?

Given these components, it is also easy to imagine the reverse process. Manipulation of a stick figure on a screen or a puppet model could result in the generation of a Benesh Movement Notation score. Calvert [Calvert80] has developed a system using a number of goniometers connected to a person used to record movement data and produce Labanotation from that input.

The design and implementation of these two interesting ideas could constitute some rather non-trivial research efforts.

5.3 Extended Notational Functionality

A great deal of work has yet to be done to extend the notational functionality of the editor. To date a substantial subset of the many Benesh signs needed to produce a ballet solo has been implemented. Yet, many items such as basic signs to specify wrist directions, finger positions, composite signs specifying travel and stage location (entry onto stage from wings), legato lines, duration brackets, etc. are not available in this version of the editor.

Benesh Movement Notation is dynamic in that the syntax of the notation is evolving to accommodate new requirements such as new styles of dance. This editor has been designed specifically with the notation of classical ballet in mind. To facilitate a much larger vocabulary, the concept of a *Sketchpad* has emerged. If the user requires a special sign not available in the current editor, an existing sign could be customised or a new one created using this facility. The Sketchpad would consist of two components, namely a library of signs and the tools required to manipulate the contents of the library.

The library might contain all the 'basic' signs provided by the editor, as well as a collection of previously created customised signs. A mechanism would be needed to peruse the library to find a specific sign. Forcing the user to scroll through a potentially large library of signs does not look like an attractive approach. Perhaps a hierarchical structure could be imposed on the library to simplify access to signs. One could also imagine a library of high-usage signs for each of a number of styles of dance. A mechanism to allow access to any one of many libraries would then be needed.

Some tool to maintain the library (add and delete signs as well as specify and maintain the hierarchy) will be needed. This could be used, for instance, to arrange that often-used signs are grouped in a standard menu to allow quick access. These functions may be provided within the editor itself or as a separate system. A mechanism for building new signs is also required. Such an editor would have the ability to select any number of signs from the library and place each in a work area individually. The ability to arbitrarily rotate each component of the new sign as well as the entire composed sign are also required. Finally, a free-hand sketch facility such as that employed to draw in-frame movement lines in ChoreoScribe is needed. When finished, the newly created sign could then be

placed in a frame as part of the score and also stored in a library of signs.

One important issue to consider here is that such a sign is not compatible with the body model concept of the editor. It is not possible to determine which limb or what staging information the composed sign is defining unless it is explicitly specified .

Another area in which further work is needed is the *spacing of frames* on a stave line. Currently a simple algorithm has been implemented to accomplish this task. More study is required to find an algorithm that produces a more aesthetically pleasing spacing of the frames. If an appropriate algorithm cannot be found or implemented, user-controlled adjustment of the placement of some frames may be required. This would be used only in those rare cases in which the existing algorithm did not produce satisfying results. The system must then be aware of which frames were placed manually to be sure to maintain the specified spacing or adjust it according to certain rules.

An objective of the design of the current version of the editor is to provide the facilities required to notate a solo ballet dance. Another desirable enhancement is the ability to notate *multiple performers* in a single score. In addition to the signs required to identify the various dancers, this also gives rise to some interesting spacing problems. Now a chronological link is needed between the frames describing one dancer and the corresponding frames of the other dancer(s). The spacing algorithm must accommodate this logical link. One can imagine a dancer standing still (resulting in frames with small data widths due to lack of movement) while another moves (notated with frames that have a larger data width). The spacing algorithm must ensure that corresponding frames, bars and score lines for different dancers appear in correct relation to one another. This influences the implementation of any operation that affects the spacing of a score line (e.g. edit a frame, add a frame, delete, move, copy a number of frames).

5.4 Enhanced Editor Functionality

Thus far, a great deal of the research effort in this project has been concentrated on computerizing the notational functionality of choreology. As this task nears completion, greater emphasis in the area of editor functionality will be required. Some of the features described here would result in improvements in the human interface as well as other software engineering aspects of the system.

In [Singh82a], a *scroll bar* is described as a method of providing feedback to indicate the length of the score and the relative position of the displayed stave lines in the score. Such a feature has been implemented in this version of the editor. To the right of the two displayed stave lines (the *display window*) a box representing the length of the length of the score is shown. The position of the display window within the entire score is illustrated by a *scroll marker* as shown in Figure 5.1.

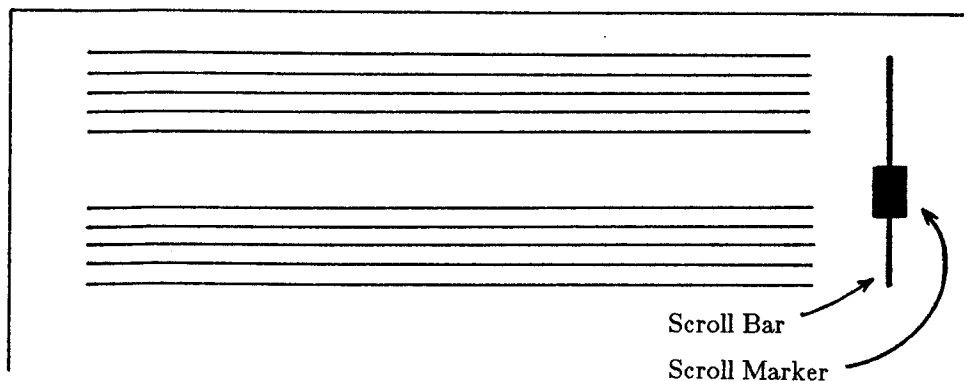


Figure 5.1 The Scroll Bar and Marker

The size of the scroll marker relative to the entire scroll bar indicates the length of the score. If the score is three stave lines in length, for instance, the scroll marker will cover two thirds of the scroll bar. If, on the other hand, the score consists of hundreds of stave lines, the position indicator will appear as a relatively small mark in the scroll bar. Two menu options, 'Next Line' and 'Previous Line', have been provided as mechanisms to scroll through the existing score. As a score grows to many stave lines, it becomes a time consuming task to scroll from the beginning of the score to the end by repeatedly selecting the Next Line command. Clearly, some mechanism to allow the user to randomly select a position within the entire score is needed. An obvious choice would be to allow the user to indicate the approximate position within the scroll bar of the stave lines of interest. This approach is common in software written for the Apple Macintosh environment. The Next Line and Previous Line commands would remain as a mechanism of 'fine-tuning' the position of the display window in the score.

A more natural way for choreologists to access parts of the score would be to provide a 'find' command. In this way, the user could identify the area of the score of interest by entering, for instance, the title of a section. The system could scan forward through the score to find a match of the string entered to all or part of a note entered in a frame. In addition, choreologists tend to mentally index scores with bar numbers. Such a facility could also be provided.

The two stave line display window appearing at the top of the screen is quite limited. At times the user would find it helpful to see more lines of the score to determine the context of a given sequence of frames. The addition of a mode in which the entire screen is used to display stave lines would prove useful, especially when scrolling through a finished score.

It has been noticed that many positions and movements in classical ballet exhibit symmetry in the cardinal sagittal plane. It is not uncommon, for instance, that when the the left arm is hanging down with the hand in front of the thigh that the right arm is also in the corresponding position reflected in the cardinal sagittal plane (as illustrated in Figure 5.2, Frame A.) Also, both arms often move in a symmetric manner such that they reach corresponding positions as in Frame B. This observation has lead to the concept of a *symmetry function*. Such an operation could be applied at a number of levels. To specify an arm (elbow and hand) or a leg (knee and foot) requires the picking and placement of one or two signs for each. Thus, up to eight signs may be used to specify the

four limbs. A Symmetry function allowing the user to specify a reflection in the cardinal sagittal plane of i) both limbs (an arm and a leg), ii) one limb or iii) a single sign (e.g. hand) would help reduce the number of actions the user must perform and increase the accuracy of the notation of a symmetric posture. Such a Symmetry function could not only apply to signs but also to free-hand drawn in-frame movement lines. The specification of such a curve involves the sketching of an approximation and possibly 'fine-tuning' the shape of the curve through control vertex manipulation. Repeating this process to accurately produce a symmetric movement line for the corresponding limb is difficult due to the nature of the interaction, since free-hand drawn curves must be able to take any shape or form. The result is at best only a close approximation to the ideal symmetric curve. Providing a function to produce such a symmetric curve automatically would simplify the interaction required and result in a more accurate and aesthetically pleasing notation. This would be a valuable improvement in the user interface of the editor.

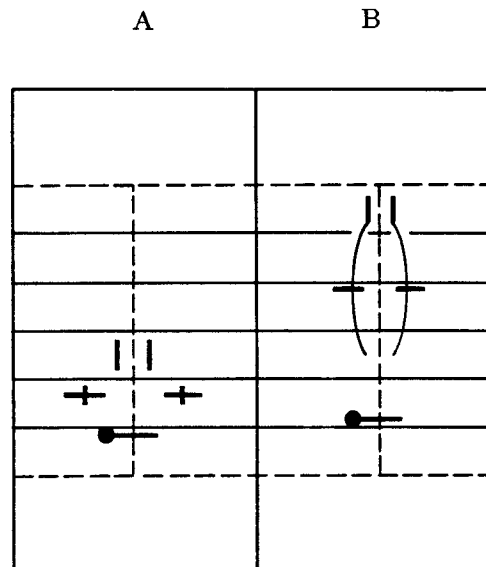


Figure 5.2 Symmetry in Notation

Another common occurrence in the process of notating a ballet is the repetition of a sequence of postures and motion. Currently, the editor uses a single *save-buffer* to store a sequence of frames to be deleted, moved, copied or simply saved for later use. Providing a number of such buffers would be useful in the notation process. Similarly, some text editors employ a numerous buffers to allow the user to work on a number of documents simultaneously [Gardner80]. In the situation here, the user would be able to read in a number of different scores and switch from one to another. A certain sequence of steps in one score may be repeated in another, for instance, and this facility would simplify the process of copying the required frames from one score to another. In this way a library of often used “macros” defining common movements, for instance, could be developed to simplify the editing process.

Related to this is the concept of a *partial archive*. As well as being able to store the entire score, providing the capability to store a certain series of stave lines may also prove helpful.

Another aspect of the notation process that has become increasingly prominent is the adherence to *constraints*. Areas in which this has become apparent include the following: i) only those menu options that are valid in the current context are visible and pick-enabled (error avoidance); ii) the placement of some signs in the frame is predefined by Benesh Movement Notation rules, whereas others may be placed in a number of valid positions and others still may be placed anywhere in the frame; and iii) body model verification of the postures and movement specified. Although it would constitute a rather fundamental change in the implementation of the editor, substantial advantages would be gained from a detailed description of ‘objects’ within the system (e.g. menus, menu options, stave lines, frames, curves, signs, symbols, etc.). A list of constraints would be compiled for each of these objects. This list would include a formal definition of the rules of the notation as well as some rules for the user interface as they apply to the objects. These constraints could then be uniformly enforced as an object is manipulated. This would lead to a more structured approach to the implementation of ChoreoScribe and would make maintenance and enhancement easier.

6 Conclusions

Benesh Movement Notation is one of a number of systems that have been developed in this century to describe human body postures and movement. It has gained fairly wide acceptance in dance companies throughout the world. The vast amount of information to be manipulated, the use of a graphical representation of the body, and the extensive use of graphical signs makes this system an ideal candidate for computerisation. This thesis describes one step in the development of a tool that can make the use of Benesh Movement Notation easier.

A number of advancements made in this version of ChoreoScribe are worth noting. One of these is a more aesthetic presentation of information (spacing of frames and lines) in the score. The use of B-splines to render curved lines and the manipulation of control vertices to modify these lines are also key contributions to the ultimate goal of the project. This also involved the development of a user-friendly interface to accomplish the required interaction. Choreologists were consulted throughout the design and development of this prototype (and especially of the user interface). As members of the intended user community for this system, their input and advice were invaluable.

The power of ChoreoScribe lies not in the initial entry of a score—substantial improvements in the user interface are still needed to make this task as easy as writing on paper— but rather in the ability to easily store, reproduce (on a screen and on paper), and modify Benesh scores.

As well as the primary objective of such a tool, which is to more easily facilitate the notation of ballets for historical preservation, a number of other applications can easily be imagined. The design of the user interface of the system has required a great deal of specification and analysis of the logical concepts of the notation. The value of this process and the resulting computer

system have been recognised as methods that can assist in the teaching of Benesh Movement Notation. The notation has also been applied to areas other than dance. In fact, any field in which body postures and motion are studied is a candidate. A description of the use of Benesh Movement Notation in anthropological research, in clinical areas and in ergonomic studies may be found in [McGuinness-Scott83]. Sports is another field in which the notation could be used. An effective notation tool would prove valuable in any application of Benesh Movement Notation.

With the emphasis on the full body model in the system described here, further possibilities also come to mind. Animation of a figure on a graphics screen is obviously one such possibility. This could also be extended to three dimensions to the control of robot movements.

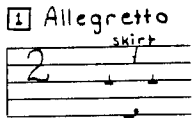

In addition to providing a tool for the application of Benesh Movement Notation, ChoreoScribe has been a test-bed for the study of a number of aspects of human interaction techniques in a computer system involving graphics. The system described here is but one step in research that can lead in a number of directions. To be truly useful in a dance production environment, work must be continued to incorporate more notation functionality in the editor. As more effective interaction devices become available, these can be studied in relation to this system. Other aspects of the software engineering and human interface of ChoreoScribe can also be studied. The application of scores produced by the editor to such areas as animation and robotics are also promising.

Appendix A - A Hand-written Benesh Score


The following is part of a hand-written Benesh score of the 2den Damesolo, Napoli, Act III notated by Rhonda Ryman. The reproduction of this extract is with the kind permission of the Institute of Choreology, London.



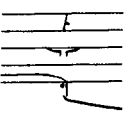
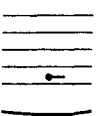

The following is a verbal description of the information found in the first stave line of this extract.

Starting Position

Frame 1	Title	2den Damesolo
2den Damesolo	Tempo	Allegretto
Allegretto	Time Signature	2 main pulses per bar
	Props	Skirt
	Head	Slightly tilted right, slightly turned left, slightly lowered
	Hands	Positioned level with the body at waist height, supporting the skirt
	Feet	Left foot flat on the floor level with the body; right foot extended behind the body, toe contacting the floor
	Location	Dancer is on the centre of centrestage
	Orientation	Dancer faces downstage right

Pick-up Beats (Frames 2 - 4)

Frame 2	Timing	Dancer holds position on count 2
	Repeats	Dotted repeat sign signals the beginning of a phrase which starts during the bar (i.e. after count 2), and which is to be repeated in

			lateral symmetry
Frame 3	Timing		Position is reached on count 2-an
	Head		Straight (no tilt, no turn)
	Feet		Dancer has stepped forward with the right foot, ending on both toes with the right foot in front of the body and the left foot behind
	Travel		Dancer moves forward toward downstage right
Frame 4	Timing		Position is reached in count 2-ti
	Feet		Dancer has stepped forward with the left foot, ending on both toes with the left foot in front of the body and the right foot behind
	Travel		Dancer continues moving forward toward downstage right
Bar 1 (Frames 5 - 8)			
Frame 5	Timing		Position is reached on count 1
	Head		Slightly tilted right, slightly turned right, slightly lowered
	Hands		Arms reach their position in front of the hips
	Feet		Right foot has stepped forward so that both feet end flat on the floor, right foot ahead of the left, and initiate a jump
Frame 6	Timing		Position is reached between counts 1 and 2
	Feet		Legs are together under the body in the air, with the right foot in front of the left
Frame 7	Timing		Position is reached on count 2
	Head		Slightly tilted left, slightly turned left, slightly lowered
	Feet		Left leg is bent with the foot flat on the ground, and the right foot contacts behind the left calf
Frame 8	Timing		Position reached on count 2-an
	Hands		Hands open sideways to waist height



Feet

Right foot has brushed the floor as it opened sideways and lifted off the floor, and the left foot initiates a jump

2^{da}n Damesolo (67)
Allegretto (musik fra "Abdallah")

skirt

skirt

skirt

skirt

skirt

skirt

skirt

skirt

skirt

skirt

Benesh Movement Notation © Rudolf Benesh, London, 1955
Production © Peter Schaufuss, Toronto, 1981

68

21

A4

f

f

f

26

c4

c4

2

• see opposite

34

PH

PH

f

f

f

f

69

Tamb.

In silence

3. Dancesolo "Jumping Solo"
 Allegretto (musik fra "Abdallah")

causes skirt to swirl around

1

Benesh Movement Notation © Rudolf Benesh, London, 1955

Choreography by August Bournonville, Copenhagen, 1841

Notation by Rhonda Ryman, Waterloo, 1988

Production © Peter Schaufuss, Toronto, 1981

Appendix B - Implemented Functions

The following is a list of Benesh signs and movement lines that have been implemented in the current version of ChoreoScribe:

- Bar Line – dashed, single or double
- Both Feet – (Cou-de-Pied) twenty signs; static and closing; five degrees of tilt
- Direction Faced – eight basic directions
- Elbow – three basic signs; automatic crossover
- Foot – four basic signs; three contact signs; automatic crossover
- Gesture Line – in-frame; free-hand sketched curve; six qualifications
- Hand – four basic signs; three contact signs; automatic crossover
- Head – tilt; turn; bend; seven positions each
- Hop/Bounce – nine signs; skimmed
- Jump Line – between frames; contracted; six qualifications; skimmed
- Knee – three basic signs; automatic crossover
- Note – any text string
- Pelvis – tilt; turn; bend; seven positions each
- Repeat – solid or dashed lines; six qualifications; nested four deep
- Rhythm Indicators – six signs
- Slide Line – between frames; contracted; six qualifications; skimmed
- Stage Location – nine basic signs; combination of two adjacent signs
- Step Line – between frames; contracted; six qualifications
- Time Signature – digits zero (0) through nine (9)
- Torso – tilt; turn; bend; seven positions each
- Travel – nine basic signs; eight basic directions
- Turn – 38 signs; clockwise or counter-clockwise; eight orientations

Appendix C - Help Menus

The following pages contain the Help menus available to the user of the editor while it is executing. These menus may be accessed in either of two manners. The first is to place the tracker over a static menu item as if to pick it. By pressing the WHITE (left-most) button on the puck, a Help menu for that particular item appears on the auxiliary terminal screen. For instance, when the Stave Edit menu is active, pressing the Help button provides a brief description of each item in the menu. The second manner in which Help information may be obtained is to simply press the white button on the puck while not pointing at a particular menu item. Depending on the state of the editor, an appropriate Help screen appears on the terminal. For example, if the Help button is pressed during the operation to specify a movement line, a Help menu for the Movement Line option appears on the terminal. In this way, if, at some point, the user is unsure of the state of the editor or what is expected next of them, pressing the Help button provides appropriate feedback.

A list of the basic functions currently available in the editor may be found in the following Help menus:

- i) H_1 - Stave Edit Menu
- ii) H_2a - Frame Edit Menu, and
- iii) H_2b - Body Part Menu.

Other menus provide more detailed descriptions of the use of the available functions.

Help - Menu 1

=====

Quit - exit Benesh Editor program
De-Archive - read a score stored on disk
- filename will be prompted for
Archive - store the current score on disk
- filename will be prompted for
Add Frame - add new frame(s) to score
- insert position may be on a new staff line
anywhere on one of the displayed lines
Edit Frame - modify one of the frames displayed
Delete - delete one or more frames
Move - delete one or more frames from current position and
place them elsewhere on displayed lines
Copy - make a copy of one or more frames elsewhere on displayed lines
Save - save one or more frames in SAVE-buffer
Put - put contents of SAVE-buffer (from last Delete, Move, Copy, Save)
on displayed lines
Next Line - scroll score to see next line
Previous - scroll score to see previous line

Quit - stop the Benesh Movement Notation Editor program
====

Note: The current score is NOT automatically stored before
the program is terminated. If the score is to be archived
the user must do so explicitly before exiting from the
editor.

De-Archive - retrieve a score stored on the disk
=====

If there are any frames displayed, the user is asked to indicate
where the stored frames are to be inserted. This may be before,
between or after the displayed lines or after any frame on these
lines.

Enter the name of the file containing the score
followed by <return>.

Archive - store current score on disk
=====

Enter the name of the file in which the score is to be stored
followed by <return>.
The file-name should contain only letters, numbers or underscores ('_').
Spaces in the name are accepted but converted to underscores.

Add Frame - add one or more frames to score
=====

If score already contains some frames, user is asked to point to the spot where new frame(s) are to be added. New frames may begin on a new line before, between or after the score lines displayed or may be inserted anywhere on the score lines displayed

'Edit mode' is then entered. The working-frame is obtained from the frame(s) immediately preceding the insert position.

Edit Frame - edit a frame of the score
=====

The user is asked to indicate which frame is to be edited. That frame is copied to the work-frame and the Frame Edit menu and Body menu appear (become active). Selection of these menu options effect modifications to the work-frame. Selection of the 'Done' option replaces the original frame with the work-frame and returns the user to the Stave Edit menu.

Delete - delete one or more frames from score
=====

The user is asked to select the beginning ('From Frame') and end ('To Frame') of the sequence of frames to be deleted. Note, the beginning and end may be the same frame in which case a single frame will be deleted. The selected sequence of frames is stored in the SAVE-buffer. That is to say that the deleted frames may now be inserted anywhere in the score using the PUT facility.

Move - move one or more frames

====

The user is asked to select the beginning ('From Frame') and end ('To Frame') of the sequence of frames to be moved. Note, the beginning and end may be the same frame in which case a single frame will be moved. The sequence of frames is deleted from the score and stored in the SAVE-buffer.

Now the user selects where the sequence of frames is to be inserted. This may be on a new score line before, between or after the displayed score lines, or the frame(s) may be inserted anywhere on the displayed lines.

NOTE: Since the SAVE-buffer contains the chosen sequence of frames, these frame(s) may be inserted elsewhere in the score again by using the PUT facility.

Copy - make a copy one or more frames

====

The user is asked to select the beginning ('From Frame') and end ('To Frame') of the sequence of frames to be copied. Note, the beginning and end may be the same frame in which case a single frame will be copied. The sequence of frames is stored in the SAVE-buffer.

Now the user selects where the sequence of frames is to be inserted. This may be on a new score line before, between or after the displayed score lines, or the frame(s) may be inserted anywhere on the displayed lines.

NOTE: Should the user wish to copy a sequence of frames to another part of the score (not displayed), the SAVE and PUT facilities should be used.

Save - save one or more frames in buffer
====

The user is asked to select the beginning ('From Frame') and end ('To Frame') of the sequence of frames to be saved. Note, the beginning and end may be the same frame in which case a single frame will be saved.

The sequence of frames is stored in the SAVE-buffer.

NOTE: Since the SAVE-buffer contains the chosen sequence of frames, these frame(s) may be inserted anywhere in the score using the PUT facility.

Put - put contents of SAVE-buffer into score
===

Select position to insert the contents of the SAVE-buffer.
Point to the position with the arrow (cursor) and press the the yellow button. The saved frame(s) may be put on a new line before, between or after the score lines displayed or after any frame on the lines displayed.

Next Line - scroll score to see next line
=====

Scroll the score to display the line immediately following the bottom score line currently displayed.

Previous - scroll score to see previous line
=====

Scroll the score to display the line immediately preceding the
first score line currently displayed.

Help - Menu 2

=====

Done - put work-frame into score and return to Menu 1
Next Line - put work-frame into score and edit next frame
which will be put at beginning of new score line
Next Frame - put work-frame into score and edit next frame
which will be put after current frame
Mus. Score - specify musical scoring information
(Time Signature, Rhythm sign, Bar-Line or Repeat sign)
Notes - specify text string
Dir. & Turn - specify Direction or Turn sign
Travel - specify Travel sign
Location - specify Stage Location sign
Mvmt. Line - specify movement line
Clear - clear the work-frame of all signs
Erase - pick a sign or movement line to be erased
Modify - pick a sign or movement line to be modified
Show/Hide - show all symbols in frame or hide those that are the same
as in the previous frame (default)

Done - enter work-frame into score & return to Menu 1
====

This menu item is used to indicate that the user is finished adding frames to the score in the current position.

The current working-frame is inserted into the score and the user is returned to Menu 1.

Next Line - add frame to score & begin new line
=====

The current working-frame is inserted into the score and editing of the next frame begins. It will be placed on a new line after the current line.

Next Frame - add frame to score
=====

The current working-frame is inserted into the score and editing of the next frame begins. It will be placed after the current frame.

Musical Scoring - specify Time Signature, Rhythm, Bar Line or Repeat sign
=====

If the work-frame is a header frame, only the Time Signature menu appears.
----- " ----- body-data frame, only the Rhythm sign menu appears.
----- " ----- scoring frame, the Bar Line menu and a Repeat sign
option appear.

Otherwise, the work-frame is a dont-know frame and all of the above options
appear.

Selection of a Time Signature, Rhythm sign or a Bar Line causes the chosen
sign to appear in the work-frame automatically in its corresponding position.

Selection of the Repeat sign option allows the user to specify a Repeat sign
in this frame.

Repeats - specify a Repeat sign in the current work-frame
=====

The user first sees a menu containing the 4 sizes of a left Repeat sign
and the 4 sizes of the right Repeat sign. One of these is to be selected.

Depending on the above selection, a menu of the 4 possible combinations
(solid & dashed lines) of the chosen sign appears. Select one of these.
It is displayed in the work-frame.

Now a menu of qualifications appear. A qualification may be selected (it is
then placed automatically in the displayed Repeat sign) or the 'Done'
option may be selected. A single or double dot identifier may also be
picked at this stage.

A menu containing the next level (two possible choices) of Repeats appears.
One of these may be selected (and it is displayed in the work-frame) and the
qualification menu appears as above.
'Done' may also be selected at this point.

When finished, if a right Repeat sign has been specified, a menu appears
allowing the choice of a number to indicate the number of times the frame(s)
are to be repeated. Select one of these or 'Done'.

Notes - specify a text string
=====

The user is asked to enter a string of characters, numbers, etc. at the keyboard. At the end of the string, press the <return> key.

Now the cursor becomes the string entered. That string is to be dragged into the working-frame and placed where desired by pressing the yellow button. Note, the beginning of the string must be placed within the working-frame area. Strings that are wider than one frame are allowed.

Direction or Turn - specify a Direction or Turn sign
=====

The user first sees a menu containing the various direction symbols.

If a Direction is to be defined, the user points to the required sign and presses the yellow button TWICE. The selected sign then appears in the work-frame.

If a Turn sign is to be defined, the user first points to the direction in which the turn starts, presses and releases the yellow button. Now, using the arrow (cursor), move either in a clockwise or counter-clockwise direction around the menu and stop at the ending position of the turn. Again, press and release the yellow button.

If the Turn defined is 7/8-ths or greater, a sub-menu of four versions of the turn sign appear. The user selects the turn sign with no number, with a 2, 3 or 'x' in its centre.

The Turn sign now appears as the cursor and is dragged into the work-frame.

Travel - specify a Travel sign in work-frame
=====

A menu of the possible travel signs first appears. The user selects one of these signs by pointing to it with the arrow (cursor) and pressing the yellow button.

A second (circular) menu appears showing the selected sign in its possible orientations. Again, a sign is selected by pointing to it and pressing the yellow button. The selected sign is then dragged into the work-frame

Stage Location - specify a stage location sign
=====

A menu of the various stage location symbols appears.

The user selects one of these symbols. A small cross bar appears below the selected symbol.

Now the user may select an adjacent location symbol in order to make a sign composed of two location symbols OR the small cross bar may be selected.

If a second location symbol is chosen, the menu disappears and composite sign appears. The cursor becomes the cross bar. It is dragged onto the composite location sign and placed by pressing the yellow button.

When the cross bar is selected, the menu disappears and the chosen symbol appears. The cursor becomes the cross bar. It is dragged onto the symbol and placed by pressing the yellow button.

Now the composed sign becomes the cursor. It is dragged into the work-frame and placed by pressing the yellow button.

Movement Line - specify a movement line

=====

The user may do one of the following:

- i) start or end a step, jump, slide or indicate a hop/bounce
 - if no step, jump or slide is in progress, one may be begun in the current work-frame
 - if a step, jump or slide is in progress, it may be ended in in current work-frame
 - specify a hop or bounce (in-frame movement)
- ii) draw a free-hand in-frame movement line.

Indicate the desired choice with the cursor and press the yellow button.

Step Line - specifying a step line

=====

Start of Line: The user is first asked to specify the position where the step line is to begin. If the step line is to be a CONTRACTED line, select a point slightly to the left of the work-frame (outside). Otherwise, (selecting a point inside the work-frame) the step line is to begin in the current frame and end in another. A dashed line appears in the work-frame and later in the stave line to indicate an unfinished locomotion line.

A menu then appears asking which foot is to do the stepping. Select one of the available items (left, right or unspecified).

End of Line: Here the user is first asked where in the work-frame the step line is to end (near a foot sign).

A menu then appears allowing the user to specify a qualification for the step line just completed or to alter the standard shape of the curve. The 'Done' option is also provided.

When specification is complete, an approximation to the step line is drawn in the work-frame.

The line appears in its entirety in the stave line when the work-frame is (re-)inserted (using Done, Next Frame or Next Line).

Jump Line - specifying a jump line
=====

Start of Line: First ensure that there is a foot sign in the current work-frame for the start (end) of the jump line.

The user is first asked to specify the position where the jump line is to begin. If the jump line is to be a CONTRACTED line, select a point slightly to the left of the work-frame (outside). Otherwise, (selecting a point inside the work-frame near a foot sign) the jump line is to begin in the current frame and end in another. A dashed line appears in the work-frame and later in the stave line to indicate an unfinished locomotion line.

End of Line: Here the user is first asked where in the work-frame the jump line is to end (near a foot sign).

A menu then appears allowing the user to specify a qualification for the jump line just completed and/or the fact that it is a 'Skim'ed jump. The 'Done' option is also provided.

When specification is complete, an approximation to the jump line is drawn in the work frame.

The line appears in its entirety in the stave line when the work-frame is (re-)inserted (using Done, Next Frame or Next Line).

Slide Line - specifying a slide line
 =====

Start of Line: First ensure that there is a foot sign in the current work-frame for the start (end) of the slide line.

The user is first asked to specify the position where the slide line is to begin. If the slide line is to be a CONTRACTED line, select a point slightly to the left of the work-frame (outside). Otherwise, (selecting a point inside the work-frame under a foot sign) the slide line is to begin in the current frame and end in another.

A menu of 3 shapes (straight line, upward curve & downward curve) appears (as well as a 'Done' option). Any number of shapes may be selected. The slide line will consist of the chosen shapes end to end. The number of shapes selected so far is displayed next to the menu.

When 'Done' is selected, a dashed line appears in the work-frame and later in the stave line to indicate an unfinished locomotion line.

End of Line: Here the user is first asked where in the work-frame the slide line is to end (under a foot sign).

A menu then appears allowing the user to specify a qualification for the slide line just completed and/or that the slide is 'Skim'ed. The 'Done' option is also provided.

When specification is complete, an approximation to the slide line is drawn in the work-frame.

The line appears in its entirety in the stave line when the work-frame is (re-)inserted (using Done, Next Frame or Next Line).

Hops and Bounces - specify a hop or bounce
 =====

Choose the required hop/bounce sign by pointing the cursor at it and pressing the yellow button.

The skim qualification may also be selected by pointing at the word 'Skim' and pressing the yellow button. If no skim is required, select 'Done'.

Get Points - get points of a free-hand drawn movement line
=====

The user is to use the cursor to sketch out the desired movement line.

Place the cursor at the beginning of the line and press the yellow button.
Release the button and draw the required line. At the end of the
movement line, press the yellow button once again.

The line sketched must follow the path the limb moves, that is, begin
and end in the direction of the movement. The line must end
near a sign in the work-frame to be valid. This may be modified
in the 'edit movement line' phase.

Qualification - specify a qualification for a free-hand drawn movement line
=====

A menu appears providing various qualifications that may be applied to a
free-hand drawn movement line.

If a qualification is chosen, the menu disappears and the cursor becomes the
chosen symbol. That symbol is dragged into the work-frame and placed
on the movement line as desired.

If the 'Done' option is selected, no qualification is associated with the
movement line in question.

Clear - clear the work-frame of all signs
=====

Remove all signs and associated movement lines from the work-frame and
allow editing the blank frame.

Note: locomotion lines that are 'in-progress' are not affected.

Erase - remove a sign or movement line
=====

The user points to a sign or movement line to be erased with the arrow (cursor).
By pressing the yellow button that item is removed from the work-frame/score.

Note: if a locomotion movement line is selected for deletion, that line
is not erased from the staff line until the staff line is redisplayed
when the work-frame is (re-)inserted into the score.

Modify - modify a sign in the work-frame or a movement line
=====

The user may point to the sign to be moved with the arrow (cursor).
By pressing the yellow button while pointing at the sign, it
becomes the the cursor. The sign may now be dragged around in
work-frame and placed by again pressing the yellow button.

The user also point to a free-hand drawn movement line in the work-frame.
A curve-modify menu appears to allow modification of the chosen line.
When finished, a qualification may be specified for the given line.

If the user chooses a locomotion line and that line is displayed in the
stave lines, that movement line is highlighted IN THE STAVE LINE. The curve-
modify menu appears to allow modification of the chosen line.
If the chosen line is split over two stave lines or does not yet appear in
the stave lines, a work area is displayed showing the shape of the line.
That line may now be edited by accessing options in the curve-modify menu
which appears.

When modification of a movement line is complete, select the 'Done' option
in the curve-modify menu.

Edit Free-Hand Line - modify a drawn movement line
=====

The user may modify the curve as follows:

- Done - no more modification of the curve required
- Show CV - show the control points used to draw curve
 - this must be done in order to modify the curve
- Hide CV - remove the control points used to draw the curve
- Move CV - to move a control point:
 - indicate the point to be moved with the cursor and press the yellow button
 - now move the control point to its new position; press the yellow button again
 - the modified curve will be re-drawn
 - continue until no more modification is required
- CV - use one less control point to draw the curve
 - this has the effect of smoothing out the curve
- + CV - use one more control point to draw the curve
 - this has the effect of making the curve fit more closely to original curve

Show/Hide - show all symbols in work-frame or only those that have changed
=====

The work-frame can be displayed in two ways:

- show only those symbols that are different from the previous frame, or
- show all symbols defined in this frame (including those that remain unchanged from the previous frame).

The default is to show only the changes.

Selection of this option alternates between these two modes of work-frame display.

Help - Body Menu

=====

Head - define head position
Torso - define torso position
Pelvis - define pelvis position
Left Arm - specify left elbow and/or left hand
Left Hand - to be implemented
Right Arm - specify right elbow and/or right hand
Right Hand - to be implemented
Left Leg - specify left knee and/or left foot
Left Foot - to be implemented
Right Leg - specify right knee and/or right foot
Right Foot - to be implemented
Both Feet - specify sign indicating position of both feet

Main Body Sign - specify a main body sign (head, pelvis, torso)
=====

When a main body sign is selected, a work area appears. If the selected body part is already defined in the working-frame, that symbol appears in the working area. Otherwise a standard symbol (no tilt, no turn, no bend) appears in the work-frame area.

Now, the long (vertical) or short (horizontal) lines may be modified by selecting them (point to desired line with arrow and press yellow button.)

If the long line is selected, the cursor and short line disappear. Moving the puck to the left or right has a corresponding affect on the long line (tilt). When the long line is in the desired position, press the yellow button. The long line stays in this position and the short line and cursor reappear.

If the short line is selected, the cursor disappears. Moving the puck up and down defines the up/down bend (moves the short line up/down.) Moving the puck to the left or right changes the short line to define turn. When the desired position is found, press the yellow button and the cursor reappears.

The above modification process continues until the 'Done' section of the work area is selected with the yellow button. The specified symbol is entered automatically in the working-frame.

Draw a Limb - specify sign(s) for a limb
=====

A menu containing basic signs for the bend of the elbow/knee and basic signs (incl. contact) of the corresponding extremity appears.

The user may select (point to with arrow and press yellow button) either a bend symbol or an extremity symbol.

If a bend sign is chosen, that sign becomes the cursor. The user drags the sign to the desired position in the work-frame and places it there by again pressing the yellow button. Now a sign to specify the extremity may be selected from the remaining menu or 'Done' may be indicated.

When a sign defining the extremity is selected, that sign becomes the cursor. The sign is dragged to the desired position in the work-frame and placed by pressing the yellow button. Now the user may choose a sign to specify a bend sign or select 'Done'.

If both a bend and extremity sign are selected the user is returned to Menu2 and the Body Menu.

Both Feet - specify a sign for both feet
=====

A menu of general first, third and fifth position signs is displayed. Select one of these by pointing to it and pressing the yellow button.

Two more menus appear:

- sub-menu of more possible signs of the type chosen above, or
- possible tilts of sign chosen.

If a tilt is chosen directly, the sign chosen above is created with the given tilt. Both the sub-menu & tilt-menu disappear.

If a sign is chosen from the sub-menu, that menu disappears and a tilt must then be chosen.

The defined sign appears and is to be dragged into the working-frame and placed in the desired position by pressing the yellow button.

Appendix D - Versatec Plotter Output

The following is an example of Benesh notation reproduced on the Versatec V-80 raster plotter (200 dots per inch).

Allegretto (musik fra "Abdallah")
skirt

The musical score is written on ten staves. The first staff begins with a large number '2' in the left margin. The notation includes various note values, rests, and slurs. Annotations include 'B4' above the second staff, 'H to P' above the eighth staff, 'A4p' above the ninth staff, and 'A4' above the tenth staff. There are also several small symbols like phi (φ) and arrows scattered throughout the score. The piece is titled 'Allegretto (musik fra "Abdallah")' with the subtitle 'skirt'.

A4p A4

A4

H A C4 C4

H A

H A A φ

HP to D PH BA

The image shows a handwritten musical score for BMN, consisting of eight systems of two staves each. The notation includes notes, rests, and various performance markings such as dynamics (p, f, t), accents, and phrasing slurs. Key annotations include 'A4p', 'A4', 'H A', 'C4', 'φ', and 'HP to D PH BA'. A double bar line with a '2' indicates a repeat or a specific measure. The score is written in a fluid, hand-drawn style.

The image shows a handwritten musical score for guitar, consisting of five staves. The notation is dense and includes various musical symbols and annotations:

- Staff 1:** A blank staff with a vertical bar line.
- Staff 2:** Contains a few notes with a dynamic marking *f* above them.
- Staff 3:** Features a complex rhythmic pattern with notes and rests. Annotations include *D φ*, *f*, and *D φ*.
- Staff 4:** Shows a melodic line with notes and rests. Annotations include *A4p*, *A4*, and *D*.
- Staff 5:** Contains notes and rests with a dynamic marking *f* below.

Throughout the score, there are numerous slurs, ties, and other musical notations. The handwriting is in black ink on a white background.

Appendix E - Imagen Laser Printer Output

The following is an example of Benesh notation reproduced on the Imagen/Canon 10/240 laser printer (240 dots per inch).

Allegretto (musik fra "Abdallah")
skirt

Handwritten musical score for "Allegretto (musik fra 'Abdallah') skirt". The score is written in 2/4 time and consists of ten systems of staves. The first system begins with a treble clef and a '2' in a box. The music is written in a single line with various notes, rests, and ornaments. The second system has a 'B4' marking. The seventh system has 'H to P' and 'H' markings. The eighth system has 'A4p' and 'A4' markings. The score includes many slurs, accents, and other performance markings.

A handwritten musical score for BMN, consisting of ten staves. The notation includes notes, rests, and various musical symbols. Annotations include:

- Staff 1: **A4p**, **A4**, **2** (measure number), **f** (dynamic), **A4** (pitch).
- Staff 2: **f** (dynamic), **A4** (pitch).
- Staff 3: **f** (dynamic), **A4** (pitch).
- Staff 4: **HA**, **C4**, **C4**, **2** (measure number), **HA** (below staff).
- Staff 5: **f** (dynamic).
- Staff 6: **HA**, **A**, ϕ (below staff).
- Staff 7: ϕ (below staff).
- Staff 8: ϕ (below staff), **f** (dynamic).
- Staff 9: **HP to D**, **PH**, **BA**, **f** (dynamic).

The image shows a handwritten musical score for four staves. The notation is dense and includes various musical symbols and annotations:

- Staff 1:** A blank staff with a vertical bar line.
- Staff 2:** Contains musical notation with a dynamic marking *f* above the staff. There are several small vertical marks (possibly accents or breath marks) above the notes.
- Staff 3:** Contains musical notation with dynamic markings *f* and *D φ* below the staff. There are also several small vertical marks below the staff.
- Staff 4:** Contains musical notation with dynamic markings *A4p* and *A4* above the staff, and a dynamic marking *f* below the staff. There are also several small vertical marks below the staff.

The score is characterized by frequent use of slurs, ties, and various dynamic markings, suggesting a complex and expressive piece of music.

Appendix F - Autologic Typesetter Output

The following is an example of Benesh notation reproduced on the Autologic APS-Micro5 typesetter (723 dots per inch).

Allegretto (musik fra "Abdallah")

skirt

2

B4

H to P

A4p

A4

A handwritten musical score for BMN, consisting of ten staves. The notation includes notes, rests, and various musical symbols. Annotations are placed above and below the staves, including chord labels and dynamic markings.

Annotations and markings include:

- Staff 1: **A4p**, **A4**, **2**, **f**
- Staff 2: **A4**, **f**
- Staff 3: **f**
- Staff 4: **H A**, **C4**, **C4**, **2**
- Staff 5: **H A**
- Staff 6: **H A**, **A**, ϕ
- Staff 7: ϕ , ϕ , **f**
- Staff 8: **H P to D**, **P H**, **B A**, **f**

Handwritten musical score for a string quartet, page 144. The score consists of five staves. The top two staves are empty. The third staff contains a melodic line with slurs and accents, starting with a 'D' and a 'phi' symbol. The fourth staff contains a bass line with notes, slurs, and a 'D' symbol. The fifth staff contains a bass line with notes, slurs, and a 'p' dynamic marking. The score includes various musical notations such as slurs, accents, and dynamic markings like 'f' and 'p'.

References

- Badler80. 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).
- Bartels83. Bartels, Richard H., John C. Beatty, and Brian A. Barsky, An Introduction to the Use of Splines in Computer Graphics, Technical Report CS-83-09, University of Waterloo, Waterloo (1983).
- Bartels84. Bartels, Richard H., John C. Beatty, Kellogg S. Booth, and Ines Hardtke, 2-D And 3-D Interactive Computer Modelling Systems, *Proceedings Graphics Interface '84*, pp. 161-165 National Research Council of Canada, (1984).
- Benesh56. Benesh, Rudolf and Joan Benesh, *An Introduction To Benesh Dance Notation*, A. & C. Black Ltd., London (1956).
- Benesh77. Benesh, Rudolf and Joan Benesh, *Reading Dance, The Birth Of Choreology*, Souvenir Press Ltd., London (1977).
- Benesh78. Benesh, Rudolf, Birth of a Language, in *Theorio to Theory, Vol. II*, , London (1978).
- Booth82. Booth, K.S. and S.A. MacKay, Techniques for Frame Buffer Animation, *Proceedings Graphics Interface '82*, pp. 213-220 NCGA, (1982).
- Breslin82. Breslin, Paul, A Powerful Interface to a High Performance Raster Graphics System, M.Math. Thesis, Department of Computer Science, University of Waterloo, Waterloo (1982).
- Brooks75. Brooks, Frederick P., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley Publications, Massachusetts (1975).

- Brown84. Brown, Ann Kipling and Monica Parker, *Dance Notation for Beginners*, Dance Books Ltd., London (1984). Labanotation & Benesh Movement Notation.
- Calvert80. 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).
- Calvert82. Calvert, T.W., J. Chapman, and A. Patla, The Simulation of Human Movement, *Proceedings Graphics Interface '82*, pp. 227-234 NCGA, (1982).
- Clarke77. Clarke, Mary and David Vaughan, *The Encyclopedia of Dance & Ballet*, Peerage Books, London (1977).
- Coffee61. Coffee, J.L., A Comparison of Vertical and Horizontal Arrangements of Alpha-numerical Materials, *Human Factors* 3 pp. 93-98 (1961).
- Cohen82. Cohen, Selma Jeanne, *Next Week, Swan Lake; Reflections on Dance and Dances*, Wesleyan University Press, Middletown, Conn. (1982).
- Cropper68. Cropper, A. and S. Evans, Ergonomics and Computer Display Design, *Computer Bulletin* 12, no. 3 pp. 94-98 (1968).
- Date81. Date, C.J., *An Introduction to Database Systems*, Addison-Wesley Publishing Company, Reading, MA (1981).
- Earl65. Earl, W.K. and J.D. Goff, Comparison of Two Data Entry Methods, *Perceptual and Motor Skills* 20 pp. 369-384 (1965).
- Edwards83. Edwards, S., Why Is Software So Hard To Use?, *BYTE*, (December 1983).
- Eshkol58. Eshkol, N. and A. Wachmann, *Movement Notation*, Weidenfeld and Nicholson, London (1958).
- Fisher84. Fisher, John M. and Joseph D. Halford, The Engineering Workstation: A User's Perspective, *Computer Graphics and Applications* 4(8) pp. 25-28 IEEE Computer Society, (August 1984).
- Foley82. Foley, J.D. and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publications, Massachusetts (1982).

- Foley82a. Foley, J.D., The Design and Implementation of User-Computer Interfaces, *Siggraph '82, Tutorial No. 7*, ACM, (July 1982).
- Foley82b. Foley, James D. and Victor L. Wallace, The Art of Natural Graphic Man-Machine Conversation, pp. 315-324 in *Tutorial: Computer Graphics*, ed. John C. Beatty and Kellogg S. Booth, IEEE Computer Society Press, Silver Spring, MD (1982).
- Foley84. Foley, James D., Victor L. Wallace, and Peggy Chan, The Human Factors of Computer Graphics Interaction Techniques, *Computer Graphics & Applications* 4(11) pp. 13-48 IEEE Computer Society, (November 1984).
- Forsey85. Forsey, David Robert, Harmony in Transposition, A Toccata for Vax and Motorola 68000, M.Math. Thesis, Department of Computer Science, University of Waterloo, Waterloo (1985).
- Gardner80. Gardner, J.A., The FRED Text Editor: Reference Manual, in *Online Documetation*, University of Waterloo, Waterloo (1980).
- Gentleman85. Gentleman, Morven, , Personal correspondence with the author, Division of Electrical Engineering, National Research Council of Canada, Ottawa (January 1985).
- Gentleman83. Gentleman, W.M., Using the Harmony Operating System, Technical Report NRC/ERB-996, Division of Electrical Engineering, National Research Council of Canada, Ottawa (December 1983).
- Grater84. Grater, Adrian, , Personal correspondence with Rhonda Ryman, Institute of Choreology, London (July 1984).
- Guest84. Guest, Ann Hutchinson, *Dance Notation, The Process of Recording Movement on Paper*, Dance Books Ltd., London (1984).
- Herbison-Evans79. 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).

- Johnson84. Johnson, Lynnea, *MacWrite*, Apple Computer, Inc., Cupertino, CA (1984).
- Joy83. Joy, W., An Introduction to Display Editing with Vi, in *UNIX Programmer's Manual, 4.2 Berkeley Software Distribution, Volume 2c*, University of California, Berkeley, Berkeley, CA (1983).
- Knuth83. Knuth, D. E., *The TEXbook*, Addison-Wesley Publishing, Reading, MA. (1983).
- Kochanek82. Kochanek, D.H.U., A Computer System for Smooth Keyframe Animation, M.Math. Thesis, Department of Computer Science, University of Waterloo, Waterloo (1982).
- Laban75. Laban, R., *Principles of Dance and Movement Notation*, Macdonald & Evans, London (1975).
- Lea83. Lea, Sylvia Carol, Fecit, A Structured Language for Describing Computer Generated Scenes, M.Math. Thesis, Department of Computer Science, University of Waterloo, Waterloo (1983).
- Lesk78. Lesk, M. E., *Typing Documents on the UNIX System Using the -ms Macros with Troff and Nroff*, Bell Laboratories, Murray Hill, NJ (1978).
- Logan77. Logan, Gene A. and Wayne C. McKinney, *Anatomic Kinesiology*, Wm. C. Brown Company Publishers, Dubuque, Iowa (1977).
- McGuiness-Scott83. McGuiness-Scott, Julia, *Movement Study and Benesh Movement Notation*, Oxford University Press, Oxford (1983).
- Miller56. 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).
- Ossanna76. Ossanna, Joseph F., *NROFF/TROFF User's Manual*, Bell Laboratories, Murray Hill, NJ (1976).
- Rupp81. Rupp, Bruce A., Visual Display Standards: A Review of Issues, *Proceedings of the SID* **22/1** pp. 63-72 (1981).

- Ryman82. Ryman, Rhonda and Adrian Grater, *Benesh Movement Notation, Elementary Solo Syllabus, Ballet Application, Lessons 1-14*, University of Waterloo & The Benesh Institute of Choreology, Waterloo & London (1982). Course Notes - Dance 241
- Ryman84. Ryman, Rhonda, *Benesh Movement Notation, Elementary Solo Syllabus, Ballet Application, Lessons 15-21 (Draft)*, University of Waterloo, Waterloo (1984). Course Notes - Dance 341
- Saunders46. Saunders, R.D., *Danscore -- The Easy Way to Write a Dance*, published by author, Hollywood (1946).
- Shneiderman84. Shneiderman, Ben, Response Time and Display Rate in Human Performance with Computers, *Computing Surveys* 16, no. 3 pp. 265-285 The Association for Computing Machinery, Inc., (September 1984).
- Shoup79. Shoup, R.G., Colour Table Animation, *Computer Graphics* 13(2) pp. 8-13 (August 1979).
- Singh82. Singh, Baldev, A Graphics Editor for Benesh Movement Notation, M.Math. Thesis, Department of Computer Science, University of Waterloo, Waterloo (1982).
- Singh82a. Singh, Baldev, J.C. Beatty, K.S. Booth, and R. Ryman, A Graphics Editor for Benesh Movement Notation, Technical Report, Department of Computer Science, University of Waterloo, Waterloo (December 1982).
- Singh83. Singh, Baldev, J.C. Beatty, K.S. Booth, and R. Ryman, A Graphics Editor for Benesh Movement Notation, *Computer Graphics* 17(3) pp. 51-62 Association for Computing Machinery, (July 1983).
- Snowberry83. Snowberry, K., Computer Display Menus, *Ergonomics* 26, no. 7 pp. 699-712 (1983).
- Tilbrook76. Tilbrook, D., A Newspaper Page Layout System, M.Sc. Thesis, Department of Computer Science, University of Toronto, Toronto (1976).
- Tombaugh85. Tombaugh, Jo W., Michael D. Arkin, and Richard F. Dillon, The Effect of VDU Text-Presentation Rate on Reading Comprehension and Reading Speed, *CHI '85 Proceedings*, pp. 1-6 The Association for Computing Machinery, Inc., (1985).

- Yamada84. Yamada, Jiro, Nagatoshi Saito, and Akio Tamara, A Low-Cost Drafting System Based on a Personal Computer, *Computer Graphics and Applications* 4(5) pp. 61-65 IEEE Computer Society, (May 1984).

