MIT IN MOTION: AN INTERACTIVE MULTIMEDIA INFORMATION RETRIEVAL SYSTEM

by

Corinne Wayshak

Submitted to the Department of
Electrical Engineering and Computer Science

In Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Science in Electrical Engineering
at the Massachusetts Institute of Technology

May 1989

Author [illegible]
Department of Electrical Engineering and Computer Science
May 22, 1989

Certified by [illegible] 5/19/89
Glorianna Davenport
Thesis Supervisor

Accepted by ______________________
Leonard A. Gould
Chairman, Department Committee on Undergraduate Thesis

MIT IN MOTION: AN INTERACTIVE MULTIMEDIA INFORMATION RETRIEVAL SYSTEM

by

Corinne Wayshak

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 1989

in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Engineering.

ABSTRACT

Multimedia environments offer the ability to incorporate visual, textual and aural material in a coherent and contiguous manner. While most systems achieve a true multimedia presentation, many fail to utilize any media other than text for active retrieval methods.

The system designed and implemented is an interactive cinematic guide to life at M.I.T. that incorporates visual and textual information for both program material and active information retrieval. The interface design, developed for an audience with limited computer background, eliminates the keyboard, and navigation throughout the system is entirely mouse driven. The program material consists of full-motion video segments of student life as well as stills of buildings on campus, in Boston, and in Cambridge; the programmed material utilizes *HyperTalk* and a Pascal-routine library for specialized features. The non-linearity of the system will allow for user-specific voyage through its contents, both visual and textual.

Thesis Supervisor: Glorianna Davenport
Title: Assistant Professor in Media Arts and Sciences

Acknowledgments

I would like to thank my thesis supervisor, Glorianna Davenport, foremost for providing me the inspiration for my own project with her interactive environment of *New Orleans in Transition, 1983-1986*. Her advice and support have given me valuable insight.

I would also like to thank Hans Peter Brondmo for allowing me to use his developmental version of the VISUAL Tools and for quickly ironing out bugs and my complaints. His enthusiasm and support in answering my many questions during the implementation of the system helped me enormously.

Many thanks go to Robert Kaynor and the MIT Planning Office for providing me with the hardware and financial support without which this project could not have been realized.

I would also like to thank Erik Schwartz for being my crew, subject, and assistant during the filming of the videodisc material, and John Wang for starting me on the right foot in *HyperCard*.

Table of Contents

ABSTRACT 2
Acknowledgments 3
Table of Contents 4
List of Figures 7
Introduction 8

1. Background 10
1.1 Film as Art and Science 10
1.1.1 Historical Perspective 10
1.1.2 Hypermedia and the Multimedia Environment 12
1.1.3 Associative and Structural Linking 13
1.1.4 Program Material Versus Programmed Material 14
1.1.5 Random-Access Image Storage 15
1.2 The Display Environment 15
1.2.1 Hardware 15
1.2.2 Software 15
1.2.3 System Overview 16

2. Approach 19
2.1 Overview and Goals 19
2.2 The Preparation of the Videodisc 20
2.2.1 Movie as Story and Data 20
2.2.1.1 Linear Narrative Film as Data 20
2.2.1.2 Observational Moviemaking 21
2.2.1.3 Movie Chunks for Non-linear Viewing 22
2.2.2 Maps and Stills as Visual Tools for Surrogate Travel 24
2.3 Program Design for Intuitive Information Retrieval 26
2.3.1 Outlining Data 26
2.3.2 Spatial Relationships 27
2.4 Central Data Structure 29

3. Functional Description 31
3.1 Overview of Scope of Implementation 31
3.1.1 The Overall Scheme of Categorization 31
3.2 Means of Navigation 32
3.2.1 Categorical Travel 33
3.2.2 Related Information Retrieval 34
3.2.3 Navigation by Maps 34
3.2.3.1 Video Information Retrieval 35
3.3 Interface Design 35
3.3.1 Categorical Card Layers 35
3.3.2 The Information Card 37
3.3.3 Navigational Icons 38
3.3.4 The Movie Strip 39

3.4 The Centralized Database 39
3.4.1 Field Information 39
3.4.2 Data Handling Routines 41

4. Design Description **43**
4.1 The Overall Organizational Structure 43
4.2 The Interactive Outline 44
4.3 Means of Navigation 45
4.3.1 Related Information 46
4.3.2 Street Map Travel 46
4.3.3 Travel by T 47
4.4 Stack Scripts 48
4.4.1 Academic, Campus, and Entertainment 48
4.4.2 Maps 48
4.5 VISUAL Tools Incorporation 49
4.5.1 Initializing 49
4.5.2 Micon Painting 50
4.5.3 Micon Buttons 50
4.6 Central Data Base 51
4.6.1 General Data Handling Functions 51
4.6.2 Prototype Lookup Function 52
4.6.3 The Sorting Routines 53
4.6.4 The Automated Fill Routines 53

5. Future Modifications **55**
5.1 Iconified Building Fronts 55
5.2 Video Linking 56

Conclusion **58**

Appendix A. "Related Information" Programs **60**
A.1 Academic Stack 60
A.2 Campus Stack 62
A.3 Entertainment Stack 64

Appendix B. Academic Stack Script **66**

Appendix C. Campus Stack Script **68**

Appendix D. Entertainment Stack Script **70**

Appendix E. Map Stack Script **72**

Appendix F. VISUAL Tools: Play Segment Buttons **77**

Appendix G. Central Data Base Stack Script **79**

Appendix H. Boolean Operation Routines **85**
H.1 Frame Add 85
H.2 Size of Table 87
H.3 Fixit 88

Appendix I. Rectangular Lookup Routine **89**

Appendix J. Sorting Programs **92**
J.1 Card Sort 92
J.2 Rectangular Coordinate Sort 94
J.3 Name Sort 97
J.4 T Rectangular Sort 99

Appendix K. Automated Fill Programs **102**
K.1 Still Frame Fill 102
K.2 Map Frame Fill 104
K.3 Map Card Fill 106
K.4 T Frame Fill 108

List of Figures

Figure 1-1: Interactive Workstation Display Environment 17
Figure 2-1: An example of a general, top, and sublevel, bottom, street map. 28
Figure 3-1: The Macintosh, top, and NTSC, bottom, monitor displays. 36
Figure 3-2: Screen shot of the Central Database 40

Introduction

As early as 1945 the idea of using source material from varied media for information retrieval and analysis was born.[1] Interactive multimedia environments have the potential to simulate real-life learning experience, for we assimilate data from a multi-faceted world around us. The potential effectiveness of a multi-media, computer controlled environment lies in the richness of presentational and navigational forms. The computer's function articulates the ways in which we make associations between different media types allowing the user to intuitively explore related material.

The advent of the optical videodisc made the prospect of rapid access to multimedia information storage a feasible reality. A CAV (Constant Angular Velocity) videodisc can store up to 54,000 uniquely addressable video frames (or thirty minutes of full-motion video.) In addition, there are two audio tracks. The visual material can be in the form of stills, full-motion video, computer generated graphics, or text, while the sound can range from music to speech.

This thesis explores concepts and issues related to interactive multimedia through the "design example"[2] of an information kiosk implemented by the author. Visual Material functions as story, data type and structural link to allow users to explore aspects of life at MIT as well as points of interest in Cambridge and Boston. Documentary footage of students as well

---

[1]Janet Fiderio, *A Grand Vision*

[2]The Multimedia Lab, Apple Computer, Inc., *Multimedia Production: A Set of Three Reports*

as computer generated maps and 35mm stills of building fronts in Cambridge and Boston and on the MIT campus were shot, edited, and mastered onto videodisc, Entitled *MIT: In Motion*, an interactive system was designed and implemented using *HyperCard* and a supplemental library of Pascal based commands that allow use of miniature digital icons and movies. These visual representations act as associative links which, through programming, become active areas for information retrieval. The author uses documentary movies to give an impression of life at MIT and incorporates maps and stills into a surrogate travel type of interaction.

Chapter 1

Background

1.1 Film as Art and Science

1.1.1 Historical Perspective

The word "film", within the context of today's culture, conjures up that artistic medium by which Bogart and Bacall have been immortalized. In recent years, the moving image has played a powerful role in shaping aspects of our lives ranging from morality to politics. Despite film's prominence as an artistic form today, its origins are in scientific experimentation and documentation. Eadweard Muybridge, the photographer who recorded live action continuously for the first time, spent five years designing his contraption that would document a race-horse's gallop to determine if at some point the race-horse lifts all four hooves off the ground. Muybridge's technical breakthroughs coupled with the discovery of the use of celluloid roll film as a base for light-sensitive emulsions enabled the Edison Laboratories to invent the Kinetograph, the first true motion picture camera.[3] Other immediate efforts to document real life included the Lumiere "films" of a sneeze and a baby being fed.

Film is inherently a technological art and as such is constantly evolving as new developments occur. "Every new development added to the cinema must, paradoxically, take it nearer and nearer to its origin...the recreation of

[3]David Cook, *A History of Narrative Film*

the world in its own image, an image unburdened by...the irreversibility of time."[4] Until recently, film has been only a linearly accessible media, that is a filmmaker would create a work with a definitive beginning, middle, and end. The structured ordered succession of frames once established could not be varied.

The advent of the optical videodisc and CD ROM removes most of the linear constraints film has. "With the introduction of optical videodiscs to the market in 1979, the idea of an augmented information environment for observational movies became feasible".[5] One side of a constant angular velocity (CAV) optical videodisc can store 54,000 still frames. This is the equivalent of one half hour of movie information.

Most videodisc players have an RS232 port, a standard input/output port used in computer communication. By connecting the player to a computer workstation with RS232 communication capabilities, any frame on the disc can be accessed. Today, access time between any two frames on a disc is less than three seconds. "The viewer becomes an active agent and computer programs serve as the intermediaries between that viewer and the film, as described in a database . . . Filmic material can be seen as a vast archipelago of episodes around which one can voyage in a more or less random fashion. Directed passage comes from particular needs, at particular time, in accordance with the idiosyncrasies of a particular viewer."[6]

---

[4]Andre Basin, *The Myth of Total Cinema*

[5]Glorianna Davenport, *New Orleans in Transition, 1983-1986: The Interactive Delivery of a Cinematic Case Study*

[6]Nicholas Negroponte, *The Impact Of Optical Videodiscs on Filmaking*

Unlike other information media, such as many of the informational video tapes that are on the market, interactive videodiscs can have the ability to personalize a presentation and to direct a viewer's attention to possible areas of interest. Videodisc technology, then, has the potential to provide an enhanced learning experience through sight and sound.

1.1.2 Hypermedia and the Multimedia Environment

The philosophy rooted in any hypermedia system can be traced back to the concept of hypertext. In its most basic form, hypertext is an interactive database management system that maintains associative links between textual information. Words become the branching nodes for the hierarchical structure. For example, if this paragraph were in a hypertext system, and a user was intrigued or confused by the term "hypermedia" in the first sentence, he would be able to retrieve information related to "hypermedia" by an activation mechanism achieved in many mouse driven systems by simply clicking on the word. He would then be presented with new textual information, which in turn would contain words that would lead to further informational branching.

Hypermedia integrates time dependent media and graphics with the concept of hypertext. The idea of combining several media types for interactive knowledge acquisition has historical roots dating back to 1945, when Vannevar Bush, President Roosevelt's science advisor, envisioned an on-line text retrieval system that contained not only post-war scientific literature but also sketches, photographs, and personal notes. The machine, called a MEMEX, would let you browse and make associative links between any two points in the library. You would then be able to traverse them at will.

[7] Although the machine Bush dreamed up was primitive, his schemes of information organization and retrieval through associative links and browsing lie at the heart of current interactive multimedia design.

1.1.3 Associative and Structural Linking

Means of information storage have significantly changed since 1945. Photocells and microfilm have given way to magnetic and optical storage media such as magnetic tape, videodisc, and CD-ROM. The capability of storing a multitude of information generates the problem of organization for retrieval. In a multimedia environment, ideas are represented by textual, aural, and visual information which must be parsed into small discrete units; each unit would then ideally represent a single concept. Attributes can then be given to the discrete units of information, and relationships between ideas can be extrapolated.

A far reaching goal of interactive systems is to mimic the brain's ability to store and retrieve information by referential links and intuitive access. Hypermedia systems, however, have no artificial intelligence, so there exists the inherent problem of the designer's value judgments being imposed on the system. To avoid a morass of meaningless and obscure connections and references, the designer must create sound underlying data models. By sensibly associating textual and visual information, the designer creates the network of links that become the mode of transportation.

Links are classified by two distinctions: associative links and structural

[7]Janet Fiderio, *A Grand Vision*

links. Associative links allow the user to intuitively explore associative material. For example, a user of a multimedia system may inquire about director Billy Wilder and discover that Wilder, who grew up in Germany and fled from Hitler, was influenced by another German born director Ernst Lubitsch. An associative link that may exist is a text excerpt about the many German artists that fled to Hollywood after the collapse of the Weimar Republic. Alternatively, a user might choose to view scenes from Wilder's and Lubitsch's work to compare visual directorial technique. The structural links are what make such retrieval of information possible. They enforce mapping between the interface and computer through use of software and hardware.

1.1.4 Program Material Versus Programmed Material

Although less clear cut, a distinction can also be made in interactive multimedia systems is between the *program* material and the *programmed* material. The former consists of the mulitmedia information itself, represented in the form of full-motion video, stills, music, textual articles, or graphics. The latter or programmed material consists of the structural links, software applications, and routines that make the system dynamic. Programming can activate discrete portions of the program material by interpreting and acting on user actions. When this occurs, the program material itself seems to be used as a browsing tool and the distinction between program and programmed material becomes cloudy. It is through creatively and coherently unifying artistic goals with scientific means that a designer builds an interactive multimedia system.

1.1.5 Random-Access Image Storage

For program material containing full motion video, the **CAV** (Constant Angular Velocity) **optical videodisc** format offers an alternative to the material linearity of videotape. Because the video signal (and accompanying audio signals) for each frame has its own addressable location on the videodisc, any frame on a disc can be located and retrieved in any order with the use of a computer controlled videodisc player. of full motion video.) Sections of these frames may be played back linearly at varied speeds or held, frame by frame.

1.2 The Display Environment

1.2.1 Hardware

A MacintoshII with the **ColorSpace II Videographics Board** manufactured by Mass Micro Systems supports a variety of video input and output devices. The board is designed around an Intel 82786 chip which contains a memory controller, a programmable display processor, and a graphics coprocessor. The board accepts any standard NTSC (National Television System Committee) signal as an input signal.[8]

1.2.2 Software

The following software accompanies the ColorSpaceII Board:[9]

- **The Desktop Video Desk Accessory** is used to control video mixing features, and to select a key color that will be transparent to video.

[8]Mass Micro Systems, Inc., *ColorSpaceII -- Pre-Production*

[9]ColorSpaceII: Programmer's Reference

- **The Digitizer Desk Accessory** digitizes incoming video signals in a variety of formats and speeds. It also performs image processing tasks including color compression, filtering, HSV color manipulation, and saving images to files or the Clipboard.

When the ColorSpaceII board is provided with an NTSC video source signal, desk accessories, can be used to digitize, store and manipulate video frames. In addition, the Digitizer Desk Accessory can be customized by adding special resources to the desk accessory file.

HyperCard is a data-oriented system, developed by Apple Computer, Inc. and distributed with all Macintosh computers, in which a developer is able to create environmental objects among which are cards, fields, and buttons. A user can also extract chunk expressions and perform keyword searches and Boolean functions. The programming language for *HyperCard* is HyperTalk. Routines which are written in other languages, such as C or Pascal, can be compiled and incorporated into HyperTalk. These specialized programs are referred to as XCMDS (eXternal CoMmanDS).

1.2.3 System Overview

The system described in this thesis is shown in Figure 1-1 and uses the following hardware and software components:

- a **MacintoshII** with a 40 meg hard drive and 5 meg RAM working under the Macintosh operating system, running
- ***HyperCard*** (version 1.2.2), and with
- a **ColorSpaceII Board**, manufactured by Mass Micro Systems, and software installed with
- a customized real-time video special effects library for the **NTSC digitizer**[10], and

[10]Developed by Alex Benenson

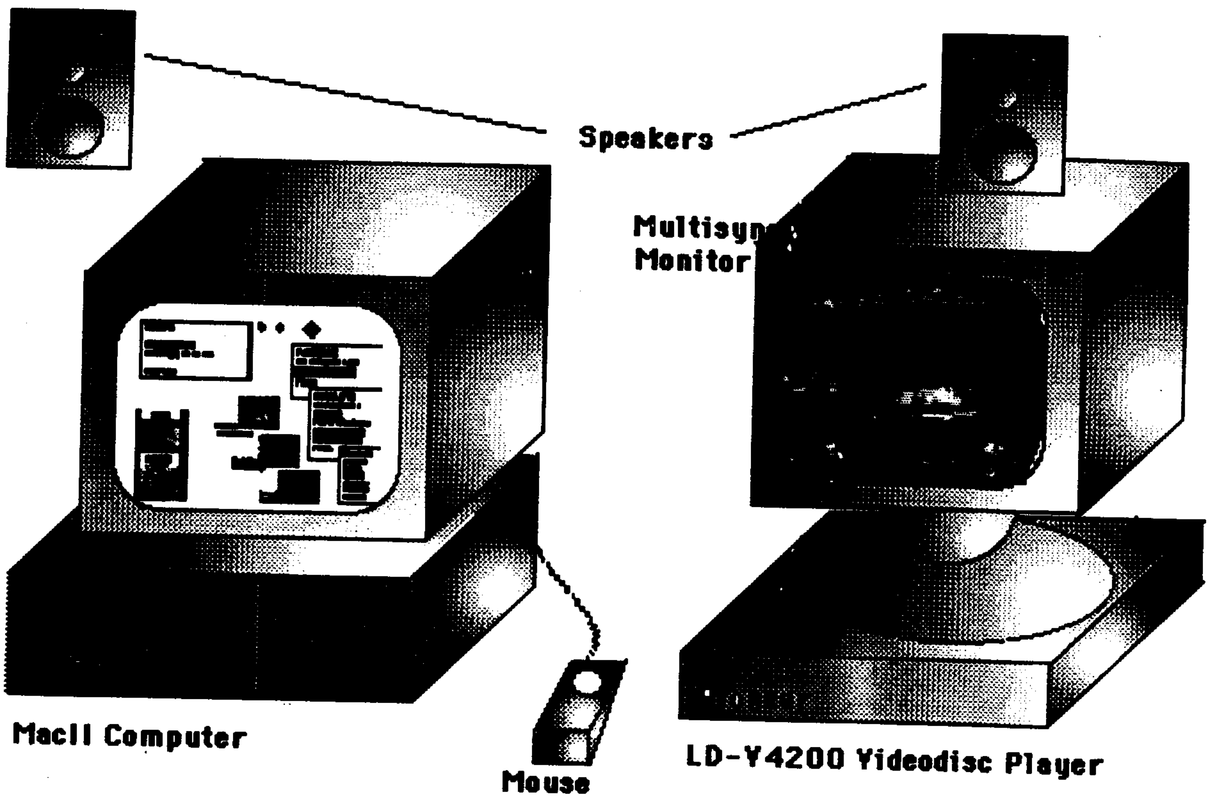


Figure 1-1: Interactive Workstation Display Environment

- a specialized set of Pascal based routines, the VISUAL Tools[11], that provide both computer control of the videodisc player and the ability to "paint" 80 x 60 color icons and digitized movies over the *HyperCard* screen,
- a **Apple 640 x 480 RGB color monitor**,
- a **Pioneer 4200 laser disc player** with computer control capabilities,
- a (13") **NEC Multisync II monitor**, and
- a pair of **Aiwa active speakers**.

[11]Written by Hans Peter Brondmo

An IBM AT with AT&T's **TrueVision Board** with NTSC output and Island Graphics **TIPS** software was used to generate titles and graphics that were transfered to videotape and mastered onto videodisc.

Chapter 2

Approach

2.1 Overview and Goals

The overall goal of this project is to create an interactive cinematic guide that would serve as an introduction to M.I.T. and its environs by incorporating visual and textual data for both entertainment purposes and active information retrieval in a kiosk environment. Because the system is developed for an audience with limited computer background, the keyboard is eliminated in the display configuration, and navigation throughout the system is entirely mouse driven. The material is organized into four main categories: **Academics, Campus Life, Entertainment**, and **Map Information.**

The implementation of this interactive project encompasses two interdependent phases: firstly, the filming, editing, and computer generation of the material to be mastered onto videodisc; and secondly, the computer programming which integrates the material into an interactive environment. From the outset, the data storage limit affected the way in which the author thought about the program material. Namely, the creation of material for the videodisc must reflect the design constraints and goal of the interactive system. Analogously design of the system is dependent on the scope and depth of the program material.

2.2 The Preparation of the Videodisc

Aimed at freshman and visitors, the material contained on the videodisc includes work and play images of student life and a series of visual navigational aids for familiarization with the cultural surround of MIT. The videodisc can easily store full motion (30fps) video, sound, still images, and computer graphics. Each media, or data type, is best suited for portraying a particular category of information. For instance, student life is best conveyed by full motion video; still photography would show student actions and interactions much less effectively. Conversely, spatial data, such as building fronts and maps, is most efficiently described by stills. Since most maps are much too detailed and complicated, local street maps, specifically designed for this system, were generated using a computer graphics program.

2.2.1 Movie as Story and Data

The creation of video material that will be computer controlled poses a set of issues not raised in traditional filmaking. One primary issue is the non-linear nature of the viewing environment. Unlike a traditional linear film where the chronological unfolding of events is fixed, in an interactive environment, segments of movie will be viewed in an order which varies from user to user. Cinematic material must therefore have the capability of being subdivided into smaller discrete units that convey meaningful information.

2.2.1.1 Linear Narrative Film as Data

Full motion material for interactive videodisc systems have taken the form of both narrative and documentary film. Members of Apple's Multi-Media Group have designed and implemented an interactive environment using *Life Story*, a narrative film that dramatizes the race for the discovery of DNA. Designed for high school students, the repurposed enactment film allows users to compare behavior of two groups of scientists. The interactions are structured to provide teachers with a resource of information from which classroom exercises can be designed.

Enacted documentary and narrative films as data present problematic issues. One difficulty is that the information presented by a narrative film is highly dependent on the knowledge and integrity of the filmaker. Another issue is the fact that the structure of the linear narrative inherently relies on the organizational thought patterns of the creator. Given that each person has his or her own way of interpreting and exploring material, this may unduly constrain the premise that an interactive environment will offer unique voyage of discovery to individual users.

2.2.1.2 Observational Moviemaking

Observational documentary filmaking offers a cinematic form where the information has less dependency on the interpretation and subjectivity of the creator. Although the filmmaker does impose a storyline and does determine **what** information is relevant to include in the film, the material presented is recorded reality. The artist has the power to distort or bias the film by presenting one-sided arguments, but he or she is powerless to change such important data as vocal tone, facial expression, or spatial relationships without digital manipulation of the recorded data.

An example of a linear documentary work used as the base of an interactive environment is *A City in Transition: New Orleans, 1983-1986*, produced, directed, and edited by Glorianna Davenport. The Cinema Verite approach encourages viewers to ask questions about the people and events which are represented. The trick is to design the interface so that viewers can move through and across issues and events as transparently as possible. Pop up menus and icon browsers are used to select new scenes or background information.[12] By presenting us with recorded events, cinematic documentation can offer insights into how people think and interact.[13]

2.2.1.3 Movie Chunks for Non-linear Viewing

The video material for *MIT: in Motion* is composed of twelve independent shorts each designed to give a student perspective on life at M.I.T and offer a glimpse of places which interest students in Boston and Cambridge. The movies are composed of documentary footage of students, and as such primarily fall under the main categories of Academics and Campus Life. The visual material in the remaining two main categories, Entertainment and Maps, takes the form of stills of building fronts. This postcard representation of Boston and Cambridge failed to render a perspective on community life around the campus, so three short segments were created which focus on showing people interacting about various landmarks. For example, the movie on Boston opens with a touristy picture of Paul Revere, across from Old North Church. A pan down reveals what people really do near such monumental historic sites: three young boys are playing baseball using the base of the statue as the catcher. As the batter hits the pitch, one of them yells, "double, second and third," and then proceeds to spit.

[12]The New Orleans system is currently being used in an introductory Urban Design and Development class at M.I.T.

[13]China Altman *Davenport Wants You as Film Pilot not Passenger*

The full motion video segments included in each category contain subject matter as follows:

- **Academics**: finals, libraries, labs, and freshmen class.
- **Campus Life**: intramural sports, late night studying, and dorms.
- **Entertainment**: Boston and Harvard Square.
- **Maps**: a ride on the T, the public transportation system.

Since all of the video pieces attempt to convey a broad perspective in a limited amount of time, namely one to two minutes, the spectrum of the material covered combined with the quick editing tends to make the segments choppy. In the *Intramural Sports* segment, ten different sports are covered in ninety seconds. One known method of smoothing abruptness at edit points is to cut on motion or gesture; the two shots on either side of the edit point become tied by completing, together, one full motion. Shots with repetitive movement or framing can be juxtaposed to smooth transitions. In the *Late Night Studying* segment, there is a shot that ends with a student, feet propped on the desk, reading a book in his lap. There is then a cut to a similarly framed shot of another student reading a book that is in the same part of the image as in the shot before it.

Another technique used to unify the shots in the video segments is to cut them to music. When shots are edited with a musical timing, the structural changes of the music support visual edits. One specific way in which the author used music was to set up a parallel content between picture and sound. The *Late Night Studying* clip is cut to song performed by Teresa Brewer and Duke Ellington entitled, "I'm Beginning to See the Light". To monopolize on the possible double meaning of the title, every time Brewer sings the word "light", there is a cut to a clock advancing in time until the final instance when a picture of dawn breaking over the Boston skyline is shown.

Two of the video segments are used as a framing device. Theoretically, when a user first approaches the system, it will be idly running the opening slide show (see Section 2.2.2.) When the user decides to interact with the system, a short video segment on *Rush and Orientation Week* will play. When the user chooses to end the session, the *Graduation* clip will play. By framing the session with the first and last events a student going through the Institute would experience, another attempt is made to give a student's perspective of MIT.

2.2.2 Maps and Stills as Visual Tools for Surrogate Travel

Early interactive video projects which implement surrogate travel concepts include the Aspen Movie Map Project, undertaken in 1979 by the Architecture Machine Group at MIT. All of the streets in Aspen were systematically filmed in stop frame motion. This footage was then mastered onto videodisc, and an interactive environment was implemented to allow a user to drive around the streets of Aspen. At an intersection, he or she could choose to continue down the same street or turn onto a new street. Overlays such as directional signals and a stop control allowed a user to directly control exploration through the city.

Another goal of *MIT: In Motion* is not so much a complete mapping of MIT and its environs, but rather an introduction of Cambridge, Boston, and the MIT campus both visually, through the use of stills of buildings, and spatially, through familiarization of local street and subway maps. To this end, approximately four hundred stills of building fronts were taken with a 35mm camera, transfered to videotape, and edited onto the footage that was mastered onto videodisc. Although textual information, such as the name of the building and its address, is associated with each picture, the stills succeed in providing visual information but fail to render a spatial sense of the buildings' location and relative positioning.

To outline the spatial relationships both between buildings and between areas of Boston and Cambridge, a series of maps were generated using AT&T's TrueVision board with Island Graphic's TIPS software. Once again, the program design is inextricably tied to the creative generation of the maps. For instance, one way of navigating at the map level is to travel to different maps bordering the map currently displayed; one example is in the map of the outer region of Harvard Square, a user can go to a map of Central Square. To visually indicate this option, a highlighted arrow pointed in the appropriate direction and labelled with "Central Square" had to be graphically included when creating the Harvard Square map. Most of the program issues relating to graphics are easily resolved once defined (See Section 2.3.2 for a more complete discussion), but several other aesthetic issues are confronted by creating the maps.

One of the difficulties in creating maps for interactive implementation is how to achieve a level of complexity, such that a substantial amount of information is shown, but to retain a level of graphic simplicity to ensure a clear and uncluttered interface design. One problem that obstructs this goal is the fact that many streets are not perfect lines but tend to curve and change direction. Besides making the map complex, bends in streets create graphic difficulties such as placing a name alongside the street. For this particular application, the map of Harvard Square presented the challenge of determining how far to simplify the map without destroying a real spatial representation of where buildings are in relation to each other.

Another visual technique used to aid the user was to make the icons for the buildings that appear on the street level maps as unique as possible. Whenever a building has a distinct architecture, the icon mimcks its shape. This technique is especially effective with the maps of the MIT campus, whose buildings range from waves to triangles. By giving icons a distinct visual signature, the system will help a user to remember icons he may have already visited.

2.3 Program Design for Intuitive Information Retrieval

The second phase of *MIT: In Motion* involved designing computer programs to integrate the material mastered onto videodisc into an interactive environment. The goal of the programming end is to create a system where information retrieval and browsing of all media types is intuitive. The model used for one of the schemes of navigation is a spatial mapping of MIT and its environs by foot and by train. In addition to understanding where he is within the system, at any time the user should also be aware of the scope of information available for perusal and the type of media he can expect to view. To achieve these goals, many navigational and interface design issues were confronted..

2.3.1 Outlining Data

One of the inherent problems in many *HyperCard* applications is the user's inability have an overview of the scope and amount of information present in the system. Depending on the way in which an application was programmed, even though a user may return to a card previously visited, he may have only traversed a small loop of information in the system.

To avoid a morass of obscure data paths, the various categories of material available for information retrieval were outlined in a hierarchical manner. For instance, the subcategory of "Restaurants" would fall under "Entertainment", one of the four main categories. Although by no means a complete mapping of all the information, this outline attempts to demonstrate many of the options available. In addition to textually presenting the scope of the system, the outline should become active. As in a hypertext environment, clicking on the text should retrieve more textual data. In this system, however, the goal is to also bring up visual data. With this basic sketch and ability to retrieve information directly from it, the user becomes more knowledgeable about the system and can efficiently extract information from specific categories.

2.3.2 Spatial Relationships

Another model used for traveling through the system is maps. Navigation by this means is explicitly tied to the reality of place, so spatial relationships between both local areas and the buildings in them can be demonstrated. To break the problem down into discrete chunks of mapping, Boston, Cambridge, and MIT were divided into sectional regions. For each of these areas, a general street map was created. Each general street map was in turn subdivided into several smaller maps which contain icons for each building whose front is stored as a still on the videodisc. (See Figure 2-1) The idea is that on the lowest level of maps, the user will be able to simulate walking down a street by visually browsing through the area and clicking on the building icons to see their fronts. The more general maps provide the user with the relationships between the streets in any given area. In addition, to

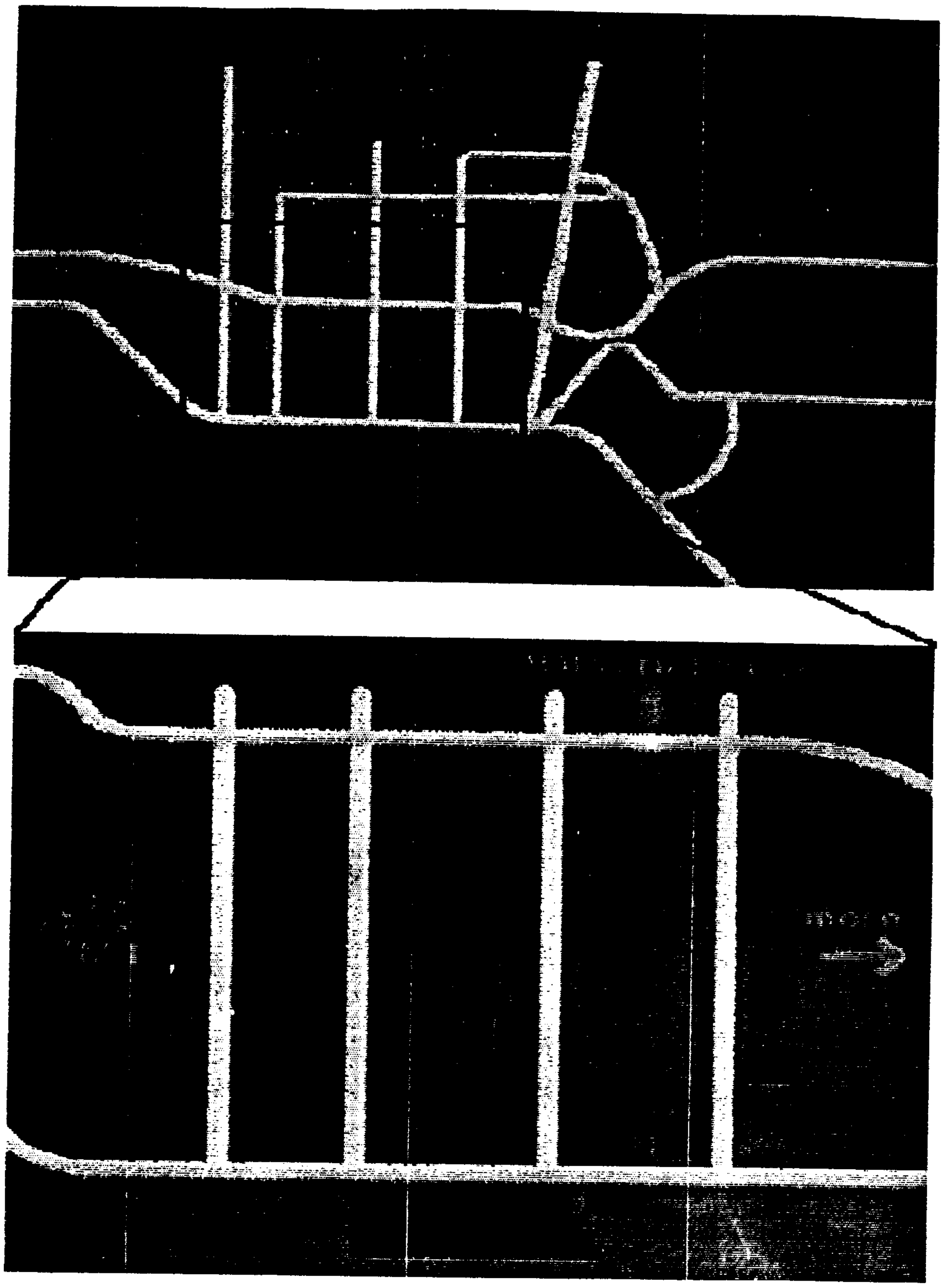


Figure 2-1: An example of a general, top, and sublevel, bottom, street map.

relate the locations of the general sections to each other, each of the subdivided maps has directions that, through programming, allow a user to go between sublevel maps of various areas located near each other. A user who is exploring a region of Harvard Square might navigate himself to the outmost region of the locale. On this sublevel map, he would find an arrow labelled "Central Square" that would navigate him to the corresponding outer edge of this sectional area of Cambridge adjacent to Harvard Square.

In addition to the street maps, a map of local stops on the public transportation system, the "T", was generated. The stops on the T-map correspond to subway stops shown on the sublevel street maps. By choosing a particular stop on the train map, a user is transported to the sublevel street map where he would arrive if he were to take the train to that particular stop. In this way, a user can explore what is around particular subway stops. Conversely, a user can choose on a sublevel map to go to the T-map by clicking on a subway stop. The subway map is then used in the programming of the system as another means to simulate travel between areas (see Section 3.2.)

2.4 Central Data Structure

The program material of the system consists of over thirty maps, approximately four hundred stills, twelve video clips, and four stacks of *HyperCard* cards. Each of these data points is interrelated in some way to many other points. For example, each building has a picture associated with it as well as textual information, and it is also located on at least one of the maps. Drawing a schematic with lines attaching linked segments to demonstrate the web of interlinking would only confuse the problem further. If the data was all hand entered and even one of the parameters change, such as receiving a new videodisc whose frame number are offset by even one frame, the time involved with updating the system would become immense as the size of the system grew.

To handle the large flow of information for this project, a centralized data base was created which could keep track of essential parameters. Discussed in detail in Section 3.4, the goal in creating such a central structure is to be able to more easily handle updates. By automating data entry into the four main stacks, the large amount of human error possible in manually entering primarily long numbers is substantially cut down. Having a central reference to all the links was a key concept in the implementation of the whole program design.

Chapter 3

Functional Description

3.1 Overview of Scope of Implementation

Once the program material is mastered onto videodisc, any piece of it can be accessed by computer within two seconds. Using *HyperCard*, its programming language HyperTalk, and a library of Pascal based routines that can be called from *HyperCard*, an interactive system that allows user-specific voyage through its visual and textual contents was designed and implemented. One of the most difficult and challenging aspects of this project was the organization of the large amount of visual and textual data and the cross-referencing and possible associations of all the data. Another large consideration was in the interface design. Since the only means of navigation through the system is by mouse, the interface must offer a satisfactory number of options to choose from while simultaneously remaining uncluttered and understandable. The following sections describe the methodological processes by which I organized the system, designed the interface for it and means of travel through it, and created a centralized data base which serves to ensure automated, thorough updates and consistency throughout the system.

3.1.1 The Overall Scheme of Categorization

As first mentioned in Section 2.1, data from all media types is categorized into four main topics: **Academics**, **Campus Life**, **Entertainment**, and **Maps**. These main categories have subcategories which in turn occasionally have a third, and maximum, level of categorization. For example, Entertainment has several categories, one of which is Restaurants which in turn has another level of categorization by type of food, such as Italian or Japanese.

Several factors lead to this categorical organization scheme. Topically, the subject matter of each of the main categories handles the four pertinent issues a perspective student or freshman must face: what is the academic workload, what are the campus and students like, what can I do for fun, and how do I get around? This fact suggested the broad divisions of material, and interface and programming considerations lead to the continuation of such an organizational scheme, for one of the main problems with *HyperCard* applications is that they generally give no overview of the material contained in the system. This problem is amplified by the elimination of the keyboard. Dividing and subdividing the material in the system into categories effectually creates a skeletal outline that can be used to give the user both a general sketch of what information is available for their perusal and one means of getting there (as described in Section 3.2.1.)

3.2 Means of Navigation

The elimination of the keyboard imposes constraints on the ways by which a user may travel throughout a *HyperCard* system. All of the means of navigation in such an environment must be programmed or scripted. The design of all four transport mechanisms transforms program material into active data types or regions that allow the user to browse and access information. The developmental goal for each navigational tool is to provide intuitive voyage through the system. Two approaches were employed to achieve this end: associative linking to suggest further topics for inquiry and spatial mapping, where the feasibility of movement is tied to the reality of a place.

3.2.1 Categorical Travel

MIT: In Motion, as an information kiosk aimed towards freshman and visitors, is designed to answer commonly asked questions such as "What can I do in town?" and "How do I get there?". The skeletal outline of the material is used as the base of one means of user navigation. Shown graphically in Section 3.3, all four of the main categories are consistently displayed throughout the system. The four labels not only give a descriptive overview but are also active sites where the user, upon clicking in the region of the word, accesses the sublevels of information associated with each main category. The listings in these subcategories are in turn active regions which may be used to access information in the system.

To illustrate the range of navigational options, let us follow the path of a new user who has asked a friend to join him for a Saturday evening "on the town." After walking up to the display, they click on "Entertainment", and a list of options appears. Clicking on one such category, "Movie Theatres", produces another level describing three types (vintage, foreign, and popular). By clicking on "Vintage", the user would get information, both visual and textual, on the Brattle Theatre in Cambridge, a vintage movie house that screens classical films. In this way, the outline of the information contained in the system functions as both associative and structural linking: textual data made active through programming provides the user with many topics to inquire into and the means of retrieving the information.

3.2.2 Related Information Retrieval

The two friends may not decide on the Brattle as the evening's form of entertainment, but want some similar suggestions. Clicking on the "Related Information" area, they request added data related to the subject matter of the information currently displayed on the monitors. In response to the users' request for more information relating to the Brattle, a movie house that screens **classical** films, textual and visual information will be retrieved about Symphony Hall, a concert hall where **classical** music is played.

3.2.3 Navigation by Maps

The two users both agree they would like to go to Symphony Hall, but they need to know how to get there. At all times in the system, two options are visible that enable transport to either the street map corresponding to the location of the most recently acquired information or to go to a map of the T, the local subway system. Clicking on the active region denoted "Street Map", the two friends bring up, on the NTSC monitor, the local street map of the appropriate area where Symphony Hall is located. Clicking on any of the icons brings up the textual and visual information associated with the building. By clicking on several buildings on the map, the two users can acquire a visual familiarity with the area before trying to find their way around the physical location.

Alternatively, a user could click on the "T map" button. This option would bring a local map of the subway system up on the NTSC monitor. Clicking on any of the stops will bring up the local street map where the stop is physically situated and graphically represented by a T subway icon. The T map is conversely accessible by clicking on any of the T subway icons located on several of the local street maps. Also included on the local street maps are directions that allow the user to travel to another local street map that borders the one currently displayed.

3.2.3.1 Video Information Retrieval

The fourth method of information retrieval and exploration is through the full-motion video segments. Each main category has associated segments that are always visibly accessible in the form of icon representations arranged in a movie strip. (See Section 3.3.4) By clicking on any of the movie icons, the appropriate short will be played on the NTSC monitor. The user in the example used throughout this section would be able to see a short piece about the Boston area. The textual information brought up on the associated card may in turn lead to another area of inquiry.

3.3 Interface Design

The interface design of the system implemented and described in this thesis is inextricably tied to navigational functionality considerations. Figure 3-1 shows an example display of the NTSC and Macintosh monitors.

3.3.1 Categorical Card Layers

As introduced in Section 3.2.1, part of the interface is devoted to aiding travel by category throughout the system. A description of the material available in outline form is consistently available on the right third of the screen. The example shown in Figure 3-1 demonstrates a card with the deepest layering of outline possible.

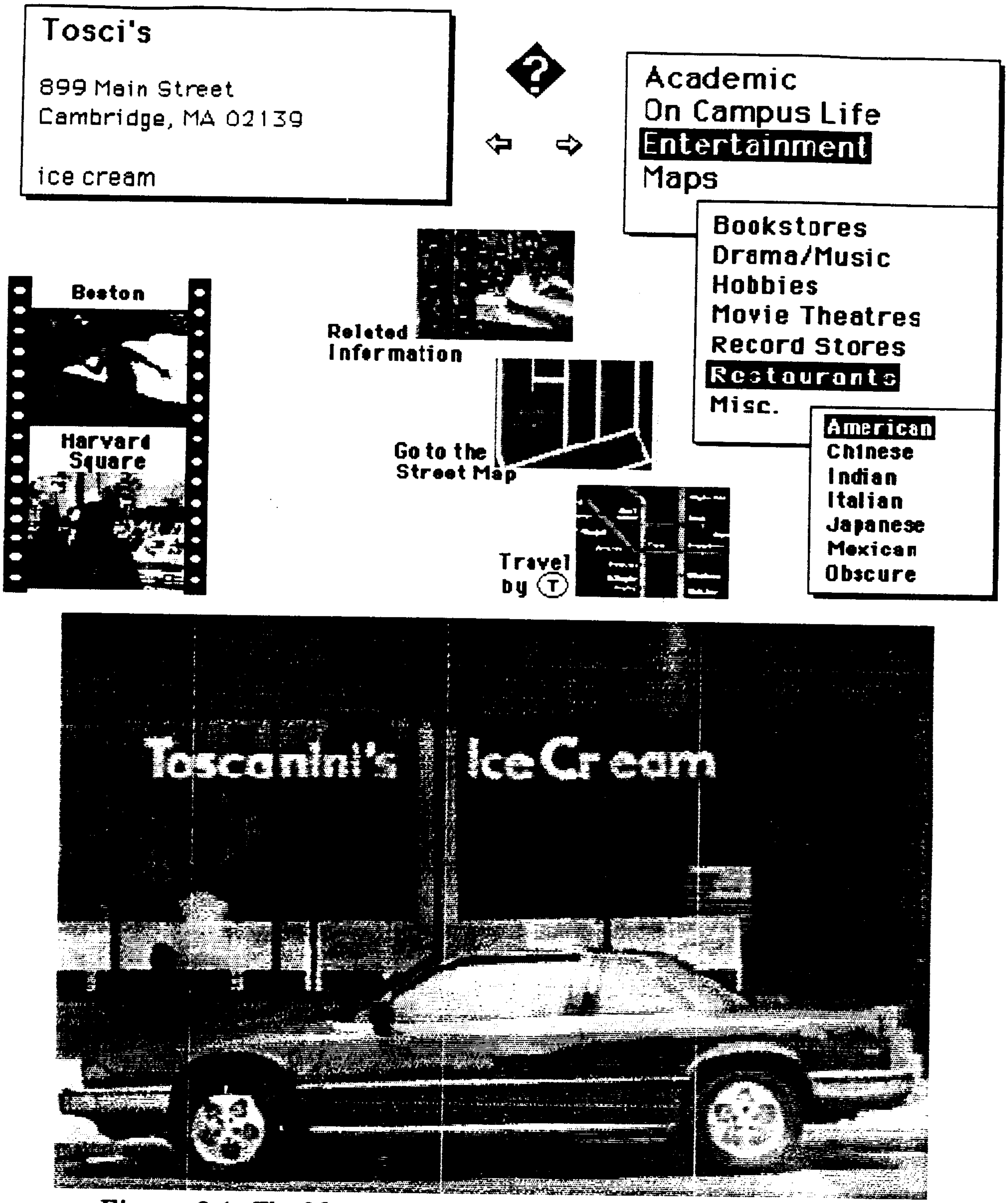


Figure 3-1: The Macintosh, top, and NTSC, bottom, monitor displays.

The top level box is always present and contains the four main categories. Clicking on any of these top four categories will cause the appropriate second level box associated with the activated topic to appear. The system contains four different second level boxes, each specifically associated with one of the four main categories. Some of the topics in the second level box will in turn lead to a third level of outline. At the lowest level, whether it is the second or third, clicking on a category will bring up textual and visual information relating to the topic chosen.

In order to know where you are within the outline, one of each category in the boxes is outlined. For the example in Figure 3-1, the categories Entertainment, Restaurants, and American are highlighted indicating the user is retrieving information about a restaurant that serves American food, in this case ice cream. Whether the user explicitly clicks on the outline or is taken to the subject matter by automated means (as described in Sections 3.2.2 and 3.2.3), the outline will highlight the area of the system the user is in.

3.3.2 The Information Card

One of the few areas of the screen not programmed to be active is the Information Card. Located at the upper left corner of the interface (See Figure 3-1), this area gives textual information; for the Entertainment section, data can include the name of a building or store, its location, phone number, and a short description of what service or merchandise is available there, while for the Academic Section, the area will reveal such information as an academic department on campus, its headquarters, and a phone number for questions.

3.3.3 Navigational Icons

The three icons in the middle of the screen (See Figure 3-1) signal three means of navigation. The top icon, labelled "Related Information", performs a keyword search using the description in the Information Card to retrieve related information to the current topic. The picture in the icon is a pair of arms reaching into a card catalog. The middle icon, a miniaturized picture of a street map, transports the user to the street map appropriate to his current location in the system. For example, if he were looking at a restaurant, clicking on the map icon would bring up the street level map where the icon for the building appears. If the user were looking at an academic department, a campus map would be retrieved. The middle icon is labelled "Street Maps" in the Entertainment section and "Campus Maps" in the Academic and Campus Life sections. The bottom icon, labeled "Take the T" brings up the map of the subway system.

Although all other aspects of the interface remain consistent, these three navigational icons vary with the main category. As stated above, there is a switch in labelling the middle icon "Street Maps" or "Campus Maps". The largest discrepancy, however, occurs in the *Maps* category. Since the skeletal outline for this section is a breakdown of the maps themselves, the "Street Maps" icon is unneeded. Additionally, since the maps do not have descriptions, the "Related Information" icon is eliminated. The "Take the T" icon is present consistently in every category.

3.3.4 The Movie Strip

The left side of the screen is devoted to a graphic representation of a film strip. In each "frame", a micon (moving icon) represents the various video clips available under the current main category. A micon is a looped playback of four seconds of digitized video. For each of the video clips, a short segment with a distinct visual signature was chosen to represent the clip. Often this visual aid is a cyclic gesture to make the transition from end to beginning of loop not so abrupt. For example, the micon for Intramural Sports contains a student hitting a tennis ball. The shot then pans with the ball to the opponent who hits the ball which is again followed back to the first student. The motion in the micon becomes a distinctive gesture which is repeatedly played.

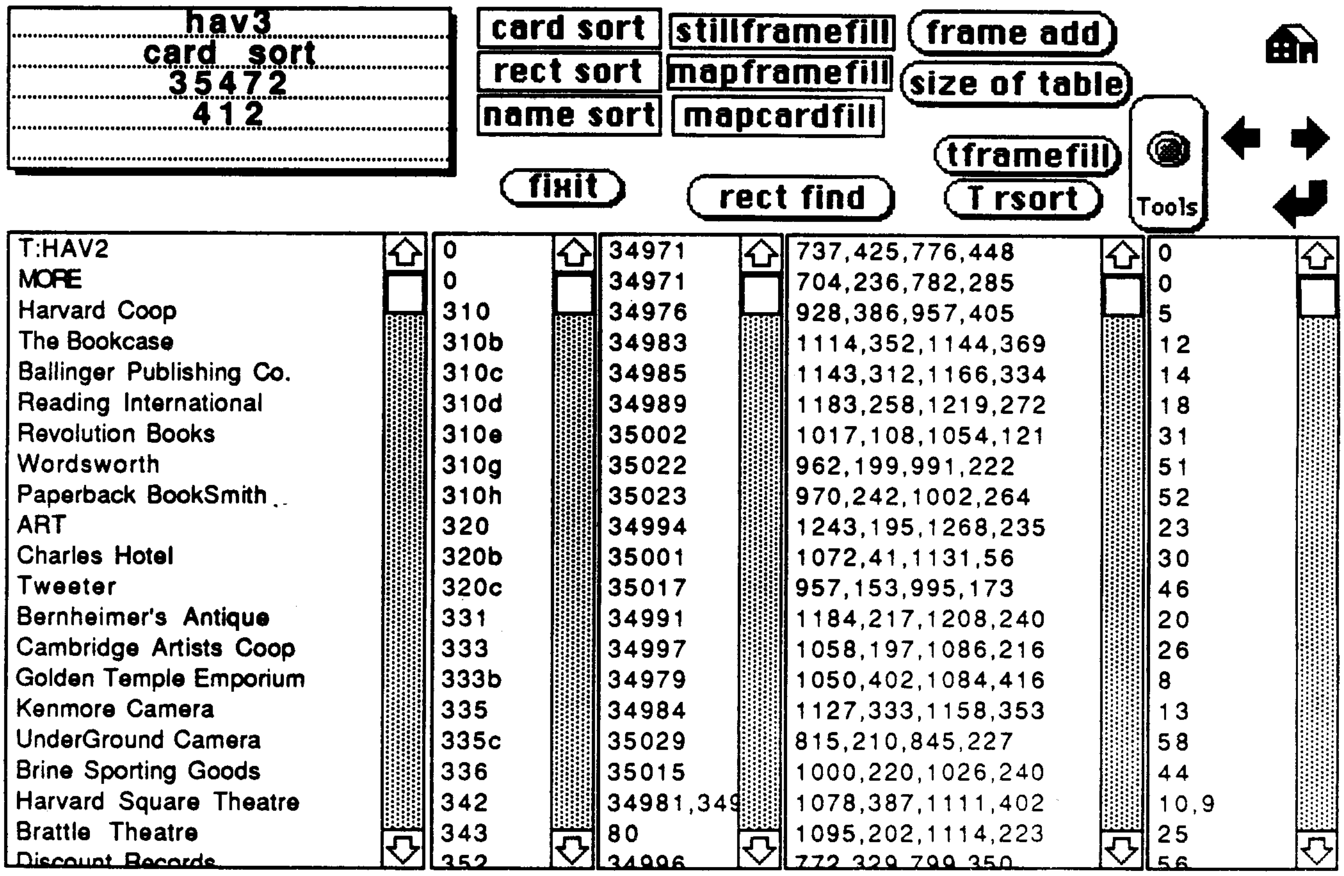
3.4 The Centralized Database

To handle the immense amount of information in the system, a central database was created with an associated set of update and sorting programs. The main organizational problem the database solves is the need to be able to assign a set of five parameters to each data chunk. The five parameters must remain linked to to each other.

3.4.1 Field Information

The general organization scheme of the database is that for each street map, there is a series of four cards in the database. Each of the four cards contains almost the identical set of information, but each is sorted by a different parameter since for different tasks and lookups, the information must be in a different order.

In the upperleft corner is a field that describes information pertaining to the map which the card is associated with. The example shown in Figure 3-2 is for the "hav3" map, a label given to the street map with building icons for the right half of the general Harvard Square map. This particular card of the series of four is sorted by cards, as shown in line two. The next line contains the frame number of the map associated with the card, 35472 in this case, while the fourth line contains the name of the card in the *Maps* stack that is associated with the map. The fifth line is used in the coordinate sorting and lookup algorithm to let the program know how many lines of data are contained in the parameter fields.



hav3
card sort
35472
412

T:HAV2	0	34971	737,425,776,448	0
MORE	0	34971	704,236,782,285	0
Harvard Coop	310	34976	928,386,957,405	5
The Bookcase	310b	34983	1114,352,1144,369	12
Ballinger Publishing Co.	310c	34985	1143,312,1166,334	14
Reading International	310d	34989	1183,258,1219,272	18
Revolution Books	310e	35002	1017,108,1054,121	31
Wordsworth	310g	35022	962,199,991,222	51
Paperback BookSmith	310h	35023	970,242,1002,264	52
ART	320	34994	1243,195,1268,235	23
Charles Hotel	320b	35001	1072,41,1131,56	30
Tweeter	320c	35017	957,153,995,173	46
Bernheimer's Antique	331	34991	1184,217,1208,240	20
Cambridge Artists Coop	333	34997	1058,197,1086,216	26
Golden Temple Emporium	333b	34979	1050,402,1084,416	8
Kenmore Camera	335	34984	1127,333,1158,353	13
UnderGround Camera	335c	35029	815,210,845,227	58
Brine Sporting Goods	336	35015	1000,220,1026,240	44
Harvard Square Theatre	342	34981,349	1078,387,1111,402	10,9
Brattle Theatre	343	80	1095,202,1114,223	25
Discount Records	352	34996	772,329,799,350	56

Figure 3-2: Screen shot of the Central Database

The fields containing parameters, numbered from left to right, for each data chunk are shown in Figure 3-2. The first field (leftmost) is the language label given to each building or active region on the map. The next field contains the name of the card associated with each line in field one. Field three contains the frame number to access on the videodisc. Field four contains the x and y relative coordinates for the upper left and lower right points that define a rectangular region for each item in field one on the map. The fifth field contains the number which describes the numerical order of each still on the videodisc. In Figure 3-2, the Harvard Coop is associated with card "**Section310**", is located at frame **34976** on the disc which is the **fifth** still on the disc, and is defined by the rectangular region whose coordinates are "**928,386,957,405**".

3.4.2 Data Handling Routines

The three sorting routines sort numerically by **card** number and **rectangular** coordinates, and alphabetically by **name**. Each of the sorting routines maintains the correspondence between the lines in each field. For example if the items are sorted by card number, if lines one and two of the field in which the card data is contained are switched, lines one and two of the rest of the fields are also interchanged.

There is also a set of automated filling programs. The **stillframefill**, **mapframefill**, and **mapcardfill** routines fill the appropriate parameters (still frame numbers, frame numbers of the associated map, and card numbers for the associated map cards) into a hidden fields on the cards in the main category stacks. These hidden numbers allow for quick access to important data needed by programs when the system is running.

The **rectfind** routine is the lookup program that figures out where each click of the mouse occurs, and if it is within the region of the NTSC monitor, also determines where the system should go next. This program, with minor modifications, is later added to the main category stacks.

The remaining programs are short routines which automate fills and sorts. The **frame add** function is used when a new videodisc is to be incorporated into the system. The program prompts you to enter the frame number of the first still. It then goes through and fills the frame access number field with the correct access numbers. The **size of table** function simply finds the number of entries on each card. This information is needed by the lookup algorithm used to identify the unique area within which the mouse clicked. The **tframefill** and **T rsort** functions are analogous the the fill and sort functions described above. Because the Tmap card in the database uses only three of the five parameter fields and adds a different field, certain lines in the program had to be changed. The **fixit** function is a short routine that had to be created when the second check disc for this project returned, and two of the frames were put on one track creating an offset of one frame after that point. The function simply went through the list of frame numbers and fixed those that were incorrectly offset.

Chapter 4

Design Description

This chapter contains a general description of the system's routines and program structure. Except for short routines, the program scripts, which are fully commented, appear in the appendices. All of the code for the system was written in *Hypertalk* and employed various commands of the VISUAL Tools software package.

4.1 The Overall Organizational Structure

To control the massive amount of information to be maintained, a highly organized numerical scheme was created. The four main categories were divided into four different stacks, each given a numerical labelling: 1) the Academic stack, 2) the campus stack, 3) the entertainment stack, and 4) the maps stack. The subcategories under each main category were assigned numbers in the same way. As data was entered into each stack, the card would be given a name reflecting the various categories under which it fell. The card for the first entry in the Japanese restaurant section is labelled "Section374"; the three represents the data is in the entertainment stack, the seven that it is a restaurant, and the four that it is specifically a Japanese restaurant. Subsequent entries under the same subcategory have a unique alphabetic sequence following the numerical code; the second Japanese restaurant would be labelled "Section374b".

4.2 The Interactive Outline

The necessity for the organizational scheme to be specifically numeric is related to the program for the interactive outline. To allow the user to travel through the information by clicking on the outline, the following program was created:

```
Field num 1 ID 3 length 436 Hidden Text-lock name "level2"
   --Find the line in this second level field
   --where the mouse just clicked. Fill this
   --in to the middle number of the card ordering
   --and go to that card.
   on mouseUp
   set cursor to watch
   get (item 2 of the clickLoc)-(item 2 of the rect of me)
   put (it div (the textHeight of me)+1) into it
   set LockScreen to true
   go to card "Section3"&it&"0"
   set cursor to hand
   set LockScreen to false
   end mouseUp
```

This specific example is from the second level of outline from the entertainment stack. The same program is used for all of the second and third level fields of the outline in all four stacks. The only change for each specific program is in the line:

```
go to card "Section3"&it&"0"
```

For the other three stacks, the "3" would be changed to the appropriate label for the stack. For third level fields in the outline, it would be the last number that needs to be modified, since while moving about a third level field, all fields above it remain constant until they themselves are clicked. The line shown above would be modified to:

```
go to card "Section36"&it
```

Since the top level in the outline will always be shown, it was implemented as a graphic pasted onto the background with buttons over each of the four words. The scripts for each of the buttons are as follows:

```
--Go to the first card of the entertainment stack.
on mouseUp
go to card 1
go to card "Section300" of stack "Entertainment"
end mouseUp

--Go to the first card of the academic stack.
on mouseUp
go to card 1
go to card "Section100" of stack "Academic"
end mouseUp

--Go to the first card of the campus stack.
on mouseUp
go to card 1
go to card "Section200" of stack "Campus"
end mouseUp

--Go to the first card of the map stack.
on mouseUp
go to card 1
go to card "Section400" of stack "Maps"
end mouseUp
```

For each card in all four stacks, the words in the outline under which the information is categorized are highlighted by setting the "hilite" property of a button covering it to true.

4.3 Means of Navigation

The three other ways of retrieving information in the system are by related information, street maps, and train travel. These three functions were implemented as scripts contained in hidden buttons underneath the digitized picture icons for each.

4.3.1 Related Information

The code for this button, shown in Appendix A, performs a systematic keyword search in all four stacks. Using the script for the button in the entertainment stack as an example, the code performs the following tasks:

- The screen is locked so that the user will not see the see various cards passing as the search is made.
- System messages are locked so that when the search starts at the card following the current one, all of the commands executed on a card opening, such as bringing up a new still associated with the card, will not occur.
- The routine grabs the first descriptive word in the last line of the information notecard at the top left corner of the screen. If the word is in an itemized list, a comma will be attached to it, so the routine checks for this and eliminates the comma if it is present.
- If the word is not found in the Entertainment stack, the Academic stack is opened to be searched. If it is found in neither, the Campus stack is opened. If the search fails, the original card is unlocked, but at any time, if a match is found, the routine takes the user to the new card and unlocks the screen and message passing.

The related information button is used in the Entertainment, Academic, and Campus stacks. Since the Map stack only contains names of the various maps available, that stack neither has the "related information" button not is searched when the button is activated from the other three stacks.

4.3.2 Street Map Travel

A program that allows the user to go to the appropriate street map for the information currently showing was implemented as a script contained in the hidden button underneath the digitized picture of a street map.

```
Bkgnd button id 19 = "steettravel"
   --Find the appropriate street map for the information
   --currently showing and go to it and its card.
   --The frame of the appropriate map will be in line
   --2 in field "links". The card of linked map will be
```

```
--in line 3 in field "links"
on mouseUp
set lockscreen to true
global frame1, card1
put item 1 of line 2 of field "links" into frame1
put item 1 of line 3 of field "links" into card1
--Mimato "EXCEPT", "VIDEO", "FRAME", frame1
go stack "Maps"
go card "Section"&card1
set lockscreen to false
end mouseUp
```

Once clicked, this program grabs the card number of the map on which the information on the current card is physically located. This card number, located in the second line of the hidden field "links", is filled in from the information in the central data base.

4.3.3 Travel by T

A program that allows the user to go to the map of the T transit system was implemented as a script contained in the hidden button underneath the digitized picture of the T map.

```
Button num 11 ID 25 length 216 name "ttravel"
--Go to the map of the T
  on mouseUp
  set lockscreen to true
  go stack "Maps"
  go cd Section4100
  --Mimato "EXCEPT", "VIDEO", "FRAME", "36417"
  set lockscreen to false
  end mouseUp
```

Unlike the code for the "streettravel" button, this program does not need to grab any information. Since the map for the T is card "Section4100", the script simply opens the "Map" stack and goes to that card.

4.4 Stack Scripts

The scripts associated with each of the four main stacks are located in Appendices B - E for Academic, Campus, Entertainment, and Maps, respectively.

4.4.1 Academic, Campus, and Entertainment

The scripts for the Academic, Campus, and Entertainment stacks are similar except for specific parameters particular to each stack. Each stack script contains code for the following functions:

- When a stack is opened, the second level category field for the stack is shown. An check for the VISUAL Tools is performed that will initialize the tools if they are not already initialized, and will paste the proper icons/micons on the screen. (See Section 4.5.2 for a fuller explanation)
- Conversely, when a stack is closed, the second level field is hidden and the micons for the stack are removed.
- When a card is opened, line one of the hidden field "links" that contains the frame number of the still associated with the card is grabbed, and that still is then retrieved and displayed on the NTSC monitor.
- A function named "playsegment" is defined that primarily serves as an alias that plays video segments.

4.4.2 Maps

The script for the Map stack is more complicated than that of the other three since it must be able to handle rectangular coordinate lookups for the NTSC monitor for each of the street maps. In addition to the functions shown in the previous section for the other three stacks, the Map stack script also contains a lookup function derived from that in the Central Data Base and described fully in Section 4.6.2. The functionality of the two are the same, and to compare differences in code, refer to Appendices I and E.

4.5 VISUAL Tools Incorporation

A set of routines was created to handle the various color icons and micons in all four main stacks.

4.5.1 Initializing

To have a standard initialization procedure for each time the system is brought up, a hidden button was created in the Entertainment stack:

```
Button num 5 ID 15 length 132 name "Tools"
  --This button is used to
  --initialize the VISUAL tools.
  on mouseUp
  Mimato "Edit"
  end mouseUp
```

After the tools have been initialized, another hidden button brings up the icons and micons for the Entertainment stack:

```
Field num 4 ID 16 length 38 name "links"
  --This button puts up all the still
  --and moving icons for the entertainment
  --stack.
  on mouseUp
  global refresh
  mimato ADD, MICON, "boston", "15,190"
  mimato ADD, MICON, "havsq", "15,277"
  mimato ADD, icon, "cardcat", "211,147"
  mimato ADD, icon, "tmap", "294,272"
  mimato ADD, icon, "streets", "251,209"
  put false into refresh
  end mouseUp
```

If at any time, the VISUAL tools are exited, the above procedure should be repeated.

4.5.2 Micon Painting

As mentioned in Section4.4.1, when a stack is opened, a check is performed to see what icons and micons must be painted. An excerpt of the Entertainment stack's script demonstrates the procedure:

```
--The micons associated with the Entertainment
--Stack are painted.
Mimato "Open", "VSEG", "Ent segments"
mimato ADD, MICON, "boston", "15,190"
mimato ADD, MICON, "havsq", "15,277"

--If the previous stack was the Academic or Campus Stack,
--then the icons do not need to be repainted. If it the
--previous stack was Maps, then repaint the icons.
if refresh is true then
mimato ADD, icon, "cardcat", "211,147"
mimato ADD, icon, "tmap", "294,272"
mimato ADD, icon, "streets", "251,209"
put false into refresh
end if
```

The first set of lines paints the micons belonging to the stack that is opened. In this example, the Boston and Harvard Square micons are displayed. The next set of lines checks to see if the icons that represent the means of travel need to be changed. They remain the same for the Academic, Campus, and Entertainment interfaces, but are different for the Map stack screen. The global variable "refresh" is used to indicate whether the icons need to be changed: the Map stack changes the status to true, so every time one of the other three stacks are opened after the Map stack, the icons are *refreshed*, and the variable is set to false again.

4.5.3 Micon Buttons

Underneath each of the micons lies an invisible button that contains a script similar to:

```
--Play the video segment on finals.
on mouseUp
playSegment "finals"
end mouseUp
```

It is the button code that activates the playing of a video segment. Appendix F contains the code for each of the segments in the four main stacks.

4.6 Central Data Base

The Central Data Base as described in Section 3.4 stores the large amount of data for the four main stacks. It also serves as a base for several programs used to update and maintain the four main stacks.

4.6.1 General Data Handling Functions

The database, being a stack itself, has a stack script (see Appendix G associated with it that contains two main programs, only one of which may be used at any given time. The program that is usually active is a version of the prototype lookup function that is described in Section 4.6.2. Its function in the database script is to test specific maps without actually leaving the map; instead of transporting the system to the card and frame parameters grabbed, the program returns them in variables that can be checked.

The second program is used when defining the rectangular regions on a map. When uncommented, the code prompts the user if he would like to enter coordinates. If the answer is yes, on the next click, the two coordinate location of the mouse relative to the upper left corner of the *HyperCard* window is automatically entered as the first two items in the first line of the "rect" field. The next click's location will be entered in the third and fourth item spot on the same line. Continuing to click will cause coordinates to be entered the line below. In this way, by systematically clicking on the upper left and then lower right corners of building icons on the maps on the NTSC monitor, a list of defined active rectangular regions is generated. When several maps are present, this process is done for the card associated with each map.

In addition to the large functions described below, several routines were quickly created to aid the adjustment and analysis of data. Shown in Appendix H these functions perform automated Boolean operations on data. The "Frame Add" and "Fixit" routines adjust frame numbers for accurate videodisc image retrieval; the "Size of Table" routine counts the number of entries in a field. Since operations of this type are usually repeated many times, it is often worthwhile to write such automation scripts.

4.6.2 Prototype Lookup Function

Appendix I contains the code for the prototype lookup function implemented in the "Rect Find" button in the Centralized Data Base. Each of the various forms of the lookup program work in the following way:

- The two coordinates of the location of the click relative to the upper right corner of the *HyperCard* window are separated and put into two variables.
- The maximum number of lines in the current table is obtained and put into another variable.
- The first and third items of the first line of the "rect" field, i.e. the two horizontal points defining a building's rectangle, are put into variables. The horizontal coordinate of the click is tested to see if it lies in the range of the first building's coordinates.
- If it does not lie in the span of the current entries, the iteration is repeated for the next line.
- If it does lie in the horizontal span, a test is performed to see if the coordinate lies in the corresponding vertical range for the building icon.
- If the coordinate's horizontal and vertical components lie in the

range, the click has been located uniquely within a building and the appropriate parameter, such as the card number to go to, is grabbed from the same line in the "card" field as the coordinate was found on in the "rect" field.

- If the coordinate's components do not lie in the current range being tested, the iteration continues until the last line of the field is reached.
- If a match is not found, the click did not take place in an active region and no action is taken. If a match is found, the routine returns the associated parameter values, and may go to a new point in the system (as is done in the Map stack script case).

4.6.3 The Sorting Routines

The code for the four sorting routines is located in Appendix J. Each program sorts the fields alphabetically or numerically by a particular parameter - card, coordinate position, or name. The function of each script is similar:

- The first line of the field for the parameter to be sorted is grabbed and compared to the line following it.
- If the first line should alphabetically or numerically appear before the line after it, the two are in proper sequence, and the next two lines are compared.
- If the lines are not in proper sequence, all the parameters for the two lines must be loaded into variables and interchanged. The line is then compared to the one before it to check if it needs to be moved nearer to the beginning of the list. Once it reaches a position where is it properly in place, the next pair of lines is compared.
- This iteration is continued until the bottom of the fields is reached, upon which time the lines of the field will be sorted by the chosen parameter.

4.6.4 The Automated Fill Routines

Code for the series of automated fill routines is shown in Appendix K. Each of the programs works by systematically going down the list of data on a card. By extracting the card number and the information to be filled from each line, the program can go to each card in one of the four main stacks and fill in the required data into hidden fields.

Chapter 5

Future Modifications

This section describes two features that were designed for the system during the process of structuring *MIT: In Motion* but were not implemented.

5.1 Iconified Building Fronts

In the current implementation, when a user chooses to go to a street map, the appropriate map is retrieved, but no indication of the exact location of the building just viewed appears. Conversely, when a user clicks on a street map, the textual and visual information is presented, but the map on the NTSC monitor is replaced by the still of the building front. The user can go back to the map by clicking on the "street map" button, but this process can become cumbersome if the user simply wants to become familiar with the buildings on a certain map.

Iconified representations of building fronts would greatly improve navigation by maps. Ideally, a user who chose to travel by the "street map" option would go to a map, either by clicking the map button or directly going to the Map category. The new design would allow a user to visually browse the buildings on a street level map. Each time he clicked on a building, a digital icon of the its front would appear next to the building. The textual information on the Macintosh screen would also change, so that even though the icon may be too small to visually recognize, the text in the information box on the *HyperCard* interface would indicate the exact name of the building. If at any time the user chose to look at the full screen picture of the building, he could simply click on the building icon.

Two obstacles exist to implementing this design for map travel. The first is that the VISUAL tools package does not provide the ability to place icons or micons on the NTSC monitor through scripting. The second problem is that the Colorspace Overlay board lacks the ability to perform real-time digitization, so each of the stills of the building fronts would have to be digitized stored. In a system, such as *MIT: In Motion*, where large numbers of stills would have to be stored, memory space quickly runs out. Both of these difficulties are relatively minor and could be overcome in the near future.

5.2 Video Linking

One of the features of the VISUAL tools package currently under development is the creation of a "linking tool" that would create a link in a video segment to another piece of data.

> The icons and micons [would be] used as link indicators. Links in video are created by dragging the chosen link indicator on top of the video and defining an in and out point for it within the segment we are linking from. On consecutive playings of the segment, the link indicator will appear spatially where it was placed and temporally when the "link in" was defined. The indicator will disappear at the "link out" point.[14]

While an indicator is visible, a user will be able to click on it to go to the related information.

With such a linking tool available, *MIT: In Motion* would have the potential to become much more dynamic. During video segments, links to relevant information could appear. For example, during the *Late Night*

---

[14]Brondmo, Davenport, *Creating and Viewing the Elastic Charles - a Hypermedia Journal*

Studying video segment, several shots of dorm rooms appear. During these shots, a **link** to the *Dorms* video segment could be used to indicate that more information about student life in dorms is available. A link indicating static information could also direct the user to the Campus Life stack subcategory of **dorms**. With the availability of a dynamic linking tool, the flow of information between different media forms would become more instinctive and natural.

Conclusion

MIT: In Motion, an interactive multi-media information retrieval system, has been designed and implemented. Through referential linking and intuitive access, the system provides the user with the ability to explore MIT, Boston, Cambridge. Full motion video provides a perspective on student life, while stills of building fronts allow a visual familiarization of MIT and its environs through extensive mapping of the areas. Visual and textual information can also be retrieved about academics, campus life, and entertainment.

To maintain and update the large amount of data, a centralized database was built and a set of automated routines was created. Using the VISUAL tools and the Color Space II Overlay board, digital icons and moving icons were incorporated into the design to become active areas. In this way, *MIT: In Motion* uses visual and textual data for both program material and interactive information retrieval.

The evolution of this project from concept to product generated a set of practical and aesthetic decisions which may prove useful to other interactive designers. The decision of what visual material to choose became inexplicitly tied to the design of the whole system. For example, the interactive environment and the half hour limitation demanded that video material have the capability of being parsed into small, meaningful sections. For *MIT: In Motion*, this fact led to the decision of creating short music videos on various aspects of student life. These segments, in turn, presented the structural issue of how to distinguish various media types. While other projects will find other solutions, the issues of granularity and media representation will face all designers.

The design of the interface offered the challenge of creating a clear and content suggestive front end with consistent symbolism. One methodology which was useful in this project was to develop a set of questions a user might have. By focusing on the demands of the user, a set of criteria for navigational means and their implementation in an understandable interface were developed.

One issue not completely resolved is how to handle large amounts of data. Although many functions were created to automate updates and manipulation of various parameters, basic components of the data had to be hand entered, an extremely time consuming process. Additionally, all links for this system had to be generated by the author. The hope is that in the near future, computer parsing algorithms will have the ability to determine visual content. Objects could then be given a set of attributes that would cross the boundaries of textual and visual media to create a truly multimedia environment.

Appendix A
"Related Information" Programs

A.1 Academic Stack

```
Button num 12 ID 27 length 2172 name "more"
  --This routine performs a keyword search across
  --the four different stacks. It picks off the
  --first word, and if that is not found, it returns
  --to the original card.

  on mouseUp
  global look1, original, new, goframe
  put word 1 of line 6 of field "info" into look1
  put the number of this card into original
  set lockscreen to true
  set lockmessages to true
  go next cd
  set lockmessages to false
  --The if statement will remove a comma if it is there.
  if last char of look1 = "," then
  delete last char of look1
  end if
  find look1 in field "info"

  if the result is empty then
  put the number of this card into new
  --If a card number is found AND it is not the same card,
  --then go to that card.
  if new <> original then
  put line 1 of field "links" into goframe
  Mimato "EXCEPT", "VIDEO", "FRAME", goframe
  set lockscreen to false
  --If a card is found AND it is the same card, go to
  --another stack.
  else
  go to stack "Entertainment"
  find look1 in field "info"
  --For each of the other 2 stacks (the map stack is not
  --included in the search), if a card is found,
  --go to the card and it's frame. If a card is not
  --found, cycle through the rest of the stacks until
```

```
--the first is reached, and then return to the original
--card.
if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false
else
go to stack "Campus"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false

else
go to stack "Academic"
go to card original
set lockscreen to false
end if
end if
end if
else

go to stack "Entertainment"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false
else
go to stack "Campus"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false
else
go to stack "Academic"
go to card original
set lockscreen to false
end if
end if
end if
end mouseUp
```

A.2 Campus Stack

```
Button num 5 ID 18 length 2172 name "more"
  --This routine performs a keyword search across
  --the four different stacks. It picks off the
  --first word, and if that is not found, it returns
  --to the original card.

  on mouseUp
  global look1, original, new, goframe
  put word 1 of line 6 of field "info" into look1
  put the number of this card into original
  set lockscreen to true
  set lockmessages to true
  go next cd
  set lockmessages to false
  --The if statement will remove a comma if it is there.
  if last char of look1 = "," then
  delete last char of look1
  end if

  find look1 in field "info"

  if the result is empty then
  put the number of this card into new
  --If a card number is found AND it is not the same card,
  --then go to that card.
  if new <> original then
  put line 1 of field "links" into goframe
  Mimato "EXCEPT", "VIDEO", "FRAME", goframe
  set lockscreen to false
  --If a card is found AND it is the same card, go to
  --another stack.
  else
  go to stack "Entertainment"
  find look1 in field "info"
  --For each of the other 2 stacks (the map stack is not
  --included in the search), if a card is found,
  --go to the card and it's frame. If a card is not
  --found, cycle through the rest of the stacks until
  --the first is reached, and then return to the original
  --card.
  if the result is empty then
  put line 1 of field "links" into goframe
  Mimato "EXCEPT", "VIDEO", "FRAME", goframe
  set lockscreen to false
```

```
else
go to stack "Academic"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false

else
go to stack "Campus"
go to card original
set lockscreen to false
end if
end if
end if
else

go to stack "Entertainment"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false
else
go to stack "Academic"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false
else
go to stack "Campus"
go to card original
set lockscreen to false
end if
end if
end if
end mouseUp
```

A.3 Entertainment Stack

```
Button num 12 ID 27 length 2172 name "more"
  --This routine performs a keyword search across
  --the four different stacks.  It picks off the
  --first word, and if that is not found, it returns
  --to the original card.

  on mouseUp
  global look1, original, new, goframe
  put word 1 of line 6 of field "info" into look1
  put the number of this card into original
  set lockscreen to true
  set lockmessages to true
  go next cd
  set lockmessages to false
  --The if statement will remove a comma if it is there.
  if last char of look1 = "," then
  delete last char of look1
  end if

  find look1 in field "info"

  if the result is empty then
  put the number of this card into new
  --If a card number is found AND it is not the same card,
  --then go to that card.
  if new <> original then
  put line 1 of field "links" into goframe
  Mimato "EXCEPT", "VIDEO", "FRAME", goframe
  set lockscreen to false
  --If a card is found AND it is the same card, go to
  --another stack.
  else
  go to stack "Academic"
  find look1 in field "info"
  --For each of the other 2 stacks (the map stack is not
  --included in the search), if a card is found,
  --go to the card and it's frame.  If a card is not
  --found, cycle through the rest of the stacks until
  --the first is reached, and then return to the original
  --card.
  if the result is empty then
  put line 1 of field "links" into goframe
  Mimato "EXCEPT", "VIDEO", "FRAME", goframe
  set lockscreen to false
```

```
else
go to stack "Campus"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false

else
go to stack "Entertainment"
go to card original
set lockscreen to false
end if
end if
end if
else

go to stack "Academic"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false
else
go to stack "Campus"
find look1 in field "info"

if the result is empty then
put line 1 of field "links" into goframe
Mimato "EXCEPT", "VIDEO", "FRAME", goframe
set lockscreen to false
else
go to stack "Entertainment"
go to card original
set lockscreen to false
end if
end if
end if
end mouseUp
```

Appendix B
Academic Stack Script

```
Stack script
 --The hidden field in upper right corner is "links"

 --The following lines remove the second level field
 --and moving icons that are associated with the Entertainment
 --Stack when a message is passed in the system that the stack
 --is closing.

 on closeStack
 --play "click" tempo 180 c5e c6
 hide bkgnd field "level2"
 --mimato remove, icon, all
 mimato remove, MICON, all
 end closeStack

 --The following lines initiate the VISUAL tools if not
 --already initiated.
 on openStack
 global isInited, refresh
 show bkgnd field "level2"
 if isInited is empty then
 Mimato "Init"
 Mimato "Config"
 put true into isInited
 end if
 --Then the micons associated with the Entertainment
 --Stack are painted.
 Mimato "Open", "VSEG", "Ent segments"
 mimato ADD, MICON, "lib", "15,132"
 mimato ADD, MICON, "finals", "15,205"
 mimato ADD, MICON, "lab", "14,280"
 mimato ADD, MICON, "class", "105,280"
 --If the previous stack was the Academic or Campus Stack,
 --then the icons do not need to be repainted.  If it the
 --previous stack was Maps, then repaint the icons.
 if refresh is true then
 mimato ADD, icon, "cardcat", "211,147"
 mimato ADD, icon, "tmap", "294,272"
 mimato ADD, icon, "streets", "251,209"
 put false into refresh
```

```
end if
end openStack

--The following lines check the hidden "link" field when
--a card opens. The frame number of the visual image
--associated with the card will be grabbed, and that frame
--will be brought up on the videodisc player.
on openCard
global frame2
put item 1 of line 1 of field "links" into frame2
if frame2>0 then
Mimato "EXCEPT", "VIDEO", "FRAME", frame2
end if
end openCard

on closeCard
end closeCard

--This is a function definition. The function "playsegment"
--is used to control video segments and appears in several
--buttons on the background.
on playSegment segmentName
global recentVList, recentCList, recentCount
if recentVList is empty then put 1 into recentCount
else put recentCount + 1 into recentCount
if recentCount is 25 then
delete last item of recentVList
delete last item of recentCList
end if
put segmentName&","&recentVList into recentVList
put name of this cd&","&recentCList into recentCList
Mimato "Play", "VSEG", segmentName
end playSegment
```

Appendix C
Campus Stack Script

```
Stack script
 --The hidden field in upper right corner is "links"

 --The following lines remove the second level field
 --and moving icons that are associated with the Entertainment
 --Stack when a message is passed in the system that the stack
 --is closing.

 on closeStack
 hide bkgnd field "level2"
 mimato remove, MICON, all
 end closeStack


 --The following lines initiate the VISUAL tools if not
 --already initiated.
 on openStack
 global isInited, refresh
 show bkgnd field "level2"
 if isInited is empty then
 Mimato "Init"
 Mimato "Config"
 put true into isInited
 end if
 --Then the micons associated with the Entertainment
 --Stack are painted.
 Mimato "Open", "VSEG", "Ent segments"
 mimato ADD, MICON, "light", "15,130"
 mimato ADD, MICON, "dorm", "15,204"
 mimato ADD, MICON, "im", "15,279"
 --If the previous stack was the Academic or Campus Stack,
 --then the icons do not need to be repainted. If it the
 --previous stack was Maps, then repaint the icons.
 if refresh is true then
 mimato ADD, icon, "cardcat", "211,147"
 mimato ADD, icon, "tmap", "294,272"
 mimato ADD, icon, "streets", "251,209"
 put false into refresh
 end if
 end openStack
```

```
--The following lines check the hidden "link" field when
--a card opens. The frame number of the visual image
--associated with the card will be grabbed, and that frame
--will be brought up on the videodisc player.
on openCard
global frame2
put item 1 of line 1 of field "links" into frame2
if frame2>0 then
Mimato "EXCEPT", "VIDEO", "FRAME", frame2
end if
end openCard

on closeCard
end closeCard

--This is a function definition. The function "playsegment"
--is used to control video segments and appears in several
--buttons on the background.
on playSegment segmentName
global recentVList, recentCList, recentCount
if recentVList is empty then put 1 into recentCount
else put recentCount + 1 into recentCount
if recentCount is 25 then
delete last item of recentVList
delete last item of recentCList
end if
put segmentName&","&recentVList into recentVList
put name of this cd&","&recentCList into recentCList
Mimato "Play", "VSEG", segmentName
end playSegment
```

Appendix D

Entertainment Stack Script

```
Stack script
 --The hidden field in upper right corner is "links"

 --The following lines remove the second level field
 --and moving icons that are associated with the Entertainment
 --Stack when a message is passed in the system that the stack
 --is closing.
 on closeStack
 hide bkgnd field "level2"
 mimato remove, micon, all
 end closeStack

 --The following lines initiate the VISUAL tools if not
 --already initiated.
 on openStack
 global isInited, refresh
 show bkgnd field "level2"
 if isInited is empty then
 Mimato "Init"
 Mimato "Config"
 put true into isInited
 end if

 --Then the micons associated with the Entertainment
 --Stack are painted.
 Mimato "Open", "VSEG", "Ent segments"
 mimato ADD, MICON, "boston", "15,190"
 mimato ADD, MICON, "havsq", "15,277"

 --If the previous stack was the Academic or Campus Stack,
 --then the icons do not need to be repainted.  If it the
 --previous stack was Maps, then repaint the icons.
 if refresh is true then
 mimato ADD, icon, "cardcat", "211,147"
 mimato ADD, icon, "tmap", "294,272"
 mimato ADD, icon, "streets", "251,209"
 put false into refresh
 end if

 end openStack
```

```
--The following lines check the hidden "link" field when
--a card opens. The frame number of the visual image
--associated with the card will be grabbed, and that frame
--will be brought up on the videodisc player.
on openCard
global frame2
put item 1 of line 1 of field "links" into frame2
if frame2>0 then
Mimato "EXCEPT", "VIDEO", "FRAME", frame2
end if
end openCard

on closeCard
end closeCard

--This is a function definition. The function "playsegment"
--is used to control video segments and appears in several
--buttons on the background.
on playSegment segmentName
global recentVList, recentCList, recentCount
if recentVList is empty then put 1 into recentCount
else put recentCount + 1 into recentCount
if recentCount is 25 then
delete last item of recentVList
delete last item of recentCList
end if
put segmentName&","&recentVList into recentVList
put name of this cd&","&recentCList into recentCList
Mimato "Play", "VSEG", segmentName
end playSegment
```

Appendix E
Map Stack Script

```
Stack script
 --The hidden field in upper right corner is "links"

 on openStack
 --The following lines initiate the VISUAL tools if not
 --already initiated.
 global isInited
 show bkgnd field "level2"
 if isInited is empty then
 Mimato "Init"
 Mimato "Config"
 put true into isInited
 end if
 Mimato "Open", "VSEG", "Ent segments"
 --Remove all of the unused icons.
 mimato remove, icon, cardcat
 mimato remove, icon, streets
 mimato remove, icon, tmap
 --Add all of the appropriate icons for the
 --Map stack.
 mimato ADD, icon, "tmap2", "263,270"
 mimato ADD, MICON, "tride", "15,276"
 end openStack


 --The following lines remove the second level field
 --and moving icons that are associated with the Entertainment
 --Stack when a message is passed in the system that the stack
 --is closing.
 on closeStack
 global refresh
 hide bkgnd field "level2"
 mimato remove, icon, all
 mimato remove, MICON, all
 --Put true into the indicator to let the next
 --stack know it will need to refresh the three
 --icons for travel.
 put true into refresh
 end closeStack
```

```
--The following lines check the hidden "link" field when
--a card opens. The frame number of the visual image
--associated with the card will be grabbed, and that frame
--will be brought up on the videodisc player.
on openCard
global frame2
put item 1 of line 1 of field "links" into frame2
if frame2>0 then
Mimato "EXCEPT", "VIDEO", "FRAME", frame2
end if
end openCard

on closeCard
end closeCard

--This is a function definition. The function "playsegment"
--is used to control video segments and appears in several
--buttons on the background.
on playSegment segmentName
global recentVList, recentCList, recentCount
if recentVList is empty then put 1 into recentCount
else put recentCount + 1 into recentCount
if recentCount is 25 then
delete last item of recentVList
delete last item of recentCList
end if
put segmentName&","&recentVList into recentVList
put name of this cd&","&recentCList into recentCList
Mimato "Play", "VSEG", segmentName
end playSegment


--The following is the lookup program. For each click of
--the mouse, the location is tested and if it is within
--the region of the colorspace screen, the building is determined
--and the system goes to the appropriate card and frame.
--The following is for the configuration when the addition of
--the card loc with the upper left colorspace coordinate = 667,14
--and the lower right coordinate = 1253,448.
on mouseDown
global rect, rect1, rect2, rect3, rect4, lookup, lookup1, lookup2
global line1, line2, i, size, index, location, imin, imax
global choice, result1, answer1, check, card1
--put the location of the click into variable check
put the clickloc into check
put item 1 of check into lookup1
--adjust for the position being relative to the upper left
```

```
--corner of the hypercard card.
put lookup1 + (item 1 of the loc of card window) into lookup1
--Check to see if the horizontal coordinate is within the
--bounds of the colorspace region.
if 667<lookup1 and lookup1<1253 then
put item 2 of check into lookup2
put lookup2 + (item 2 of the loc of card window) into lookup2
--If it is within the bounds of the colorspace region,
--then check to see if the vertical coordinate is also
--in the vertical bounds of the colorspace board.
if 14<lookup2 and lookup2<448 then
put line 2 of field "links" into size
if size is empty then
--For debugging purposes to ensure all maps have the
--size of the lookup table in line 2 of field "links"
--uncomment the following line:
--answer "no table size specified in "links" field" with "ok"
exit mouseDown
end if
--Put the maximum and minimum line numbers of field
--entries into imin and imax.
put 1 into imin
put size into imax
put 1 into i
--Put the horizontal coordinates from the hidden field
--"rect" on the map stack background into variables.
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
--Keep repeating while still in the possible range
--of field entries.
repeat while i<=imax
--If the first coordinate is less than the first coordinate
--in the lookup table, there is nothing to be done.
if lookup1<rect1 then
put 0 into answer1
exit repeat

else
--If the first coordinate is between the two horizontal
--coordinates that define the area of a building, put
--the vertical coordinates of the building into variables.
if rect1<=lookup1 and lookup1<=rect3 then
put item 2 of line i of field "rect" into rect2
put item 4 of line i of field "rect" into rect4

--If the vertical coordinate is also between the
--vertical coordinates that define a building,
--the building has been found, so grab the card
--number of the building from the same line in
--the "card" field, also hidden on the map stack
--background.
if rect2<=lookup2 and lookup2<=rect4 then
put line i of field "card" into card1
exit repeat


else
--The vertical coordinate does not lie in the
--span of the current entries in the field, so
--go to the next line and try again.
put i+1 into i
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
end if



else
--The horizontal coordinate does not lie in the
--span of the current entries in the field, so
--go to the next line and try again.
put i+1 into i
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
end if
end if
end repeat

else
--The vertical coordinate does not lie in the range of
--possible vertical coordinates. For debugging purposes
--uncomment the following line:
--play boing boing boing
exit mouseDown
end if
else
--The horizontal coordinate does not lie in the range of
--possible horizontal coordinates. For debugging purposes
--uncomment the following line:
--play boing boing
exit mouseDown
end if

--When and if a card number is retrieved, find the appropriate
```

```
--stack to go to be the first number of the card.  Go to the
--stack, and then go to the appropriate card.
if char 1 of card1 = 1 then
go card "Section"&card1 of stack "Academic"
else

if char 1 of card1 = 2 then
go card "Section"&card1 of stack "Campus"
end if

if char 1 of card1 = 3 then
go card "Section"&card1 of stack "Entertainment"
end if

if char 1 of card1 = 4 then
go card "Section"&card1 of stack "Maps"
end if

end if

--For debugging purposes, place this song in the script at a
--trouble spot to figure out what's going on.
--twinkle, twinkle
--play "boing" tempo 200 "a a e5 e f# f# eh"
--dq d c# c# b4 b aw"

--If the coordinate was within the bounds of the area, but
--was not within a specific building, then nothing is to be
--done.
if i=imax+1 then
put inactive into result1
--As an aid in debugging, uncomment the following lines to
--know or verify why nothing happened.
--doom march
--play "boing" tempo 500 "ch. ch c ch. ebh d dh c ch b3 c4w"
exit mouseDown
end if

--Uncomment the following line to help let you know if things
--went as planned.
--play "boing"
end mouseDown
```

Appendix F

VISUAL Tools: Play Segment Buttons

For the Academic Stack:

```
--Play the video segment on finals.
on mouseUp
playSegment "finals"
end mouseUp

--Play the video segment on labs.
on mouseUp
playSegment "lab"
end mouseUp

--Play the video segment on libraries.
on mouseUp
playSegment "lib"
end mouseUp

--Play the video segment on freshman classes.
on mouseUp
playSegment "class"
end mouseUp
```

For the Campus Stack:

```
--Play the video segment on dorms.
on mouseUp
playSegment "dorm"
end mouseUp

--Play the video segment on intramural sports.
on mouseUp
playSegment "im"
end mouseUp

--Play the video segment on late night studying.
on mouseUp
playSegment "light"
end mouseUp
```

For the Entertainment Stack:

```
--Play the video segment on Boston.
```

```
on mouseUp
playSegment "boston"
end mouseUp

--Play the video segment on Harvard Square.
on mouseUp
playSegment "havsq"
end mouseUp
```

For the Maps Stack:

```
--Play the video segment on the T.
on mouseUp
playSegment "tride"
end mouseUp
```

Appendix G

Central Data Base Stack Script

```
Stack script
 -- The following two commented blocks are used to get
 --the summation of the position of the card and the
 --uppper left and bottom right clicks on a building
 --located on a map on the videodisk. When uncommented, the
 --program will sequentially enter the upper left and lower
 --right coordinates into a line and then move to the next line

 --****IMPORTANT***line 1 of field "map" must always
 --be the end of the card name.
 --For Example: for chav1, rhav1, nhav1, and hav1,
 -- the title of field map must be HAV1 !!!!!!!!!!!!!!

 --on openCard
 answer "find loc" with "yes" or "no"
 set lockScreen to True
 if it is "no" then
 set lockScreen to False
 exit openCard
 else if it is "yes" then
 global z, c, ind
 put 0 into z
 put 0 into c
 put true into ind
 end if
 set lockScreen to False
 --end openCard

 --on MouseDown
 global c, z, color, hyper, x, y, co1, co2, ind
 if ind is true then
 global mouseState
 --The next two lines do not need to be umcommented...
 --They are from when the "on idle" needed to be used
 --because the VISUAL tools could not pass a mouseDown.
 --if mouseState is empty then put up into mouseState
 --get the mouse
 --if it is down and mouseState is up then
 if z = 0 then put 1 into z
 if c = 0 then put 1 into c
```

```
--put the mouseloc into color
put the clickloc into color
put the loc of cd window into hyper
put item 1 of hyper into x
put item 2 of hyper into y
put item 1 of color into x1
put item 2 of color into y1
put x + x1 into co1
put y + y1 into co2
if c = 1 then
put co1 into item 1 of line z of field "rect"
put co2 into item 2 of line z of field "rect"
put c+1 into c
else
put co1 into item 3 of line z of field "rect"
put co2 into item 4 of line z of field "rect"
put 1 into c
put z+1 into z
end if
end if
--end if
--put it into mouseState
--end mousedown

--The program below is almost identical to that in
--the map stack's script. It is used here to test and
--debug problems with the maps.
--The following is the lookup program. For each click of
--the mouse, the location is tested and if it is within
--the region of the colorspace screen, the building is determined
--and the system goes to the appropriate card and frame.
--The following is for the configuration when the addition of
--the card loc with the upper left colorspace coordinate = 667,14
--and the lower right coordinate = 1253,448.
on mouseDown
global rect, rect1, rect2, rect3, rect4, lookup, lookup1, lookup2
global line1, line2, i, size, index, location, imin, imax
global choice, result1, answer1, check, card1
--put the location of the click into variable check
--adjust for the position being relative to the upper left
--corner of the hypercard card.
put the clickloc into check
put item 1 of check into lookup1
put lookup1 + (item 1 of the loc of card window) into lookup1
--Check to see if the horizontal coordinate is within the
--bounds of the colorspace region.
if 667<lookup1 and lookup1<1253 then
```

```
put item 2 of check into lookup2
put lookup2 + (item 2 of the loc of card window) into lookup2
--If it is within the bounds of the colorspace region,
--then check to see if the vertical coordinate is also
--in the vertical bounds of the colorspace board.
if 14<lookup2 and lookup2<448 then
put line 5 of field "map" into size
--For debugging purposes to ensure all maps have the
--size of the lookup table in line 2 of field "links"
--uncomment the following line:
--answer "no table size specified in "links" field" with "ok"
--if size is empty then
-- answer "no table size specified in map field"
-- play "boing" c6 c7
-- exit mouseDown
-- end if
--Put the maximum and minimum line numbers of field
--entries into imin and imax.
put 1 into imin
put size into imax
put 1 into i
--Put the horizontal coordinates from the hidden field
--"rect" on the map stack background into variables.
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
--Keep repeating while still in the possible range
--of field entries.
repeat while i<=imax
--If the first coordinate is less than the first coordinate
--in the lookup table, there is nothing to be done.
if lookup1<rect1 then
put 0 into answer1
exit repeat

else
--If the first coordinate is between the two horizontal
--coordinates that define the area of a building, put
--the vertical coordinates of the building into variables.
if rect1<=lookup1 and lookup1<=rect3 then
put item 2 of line i of field "rect" into rect2
put item 4 of line i of field "rect" into rect4

--If the vertical coordinate is also between the
--vertical coordinates that define a building,
--the building has been found, so grab the card
--number of the building from the same line in
--the "card" field, also hidden on the map stack
```

```
--background.
if rect2<=lookup2 and lookup2<=rect4 then
put line i of field "frame" into answer1
put line i of field "card" into card1
exit repeat

else
--The vertical coordinate does not lie in the
--span of the current entries in the field, so
--go to the next line and try again.
put i+1 into i
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
end if



else
--The horizontal coordinate does not lie in the
--span of the current entries in the field, so
--go to the next line and try again.
put i+1 into i
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
end if
end if
end repeat
else
--The vertical coordinate does not lie in the range of
--possible vertical coordinates. For debugging purposes
--uncomment the following line:
play boing boing boing
exit mouseDown
end if
else
--The horizontal coordinate does not lie in the range of
--possible horizontal coordinates. For debugging purposes
--uncomment the following line:
play boing boing
exit mouseDown
end if
--if answer1>34972 then
--Mimato "EXCEPT", "VIDEO", "FRAME", answer1

--When and if a card number is retrieved, find the appropriate
--stack to go to be the first number of the card. Go to the
--stack, and then go to the appropriate card.
```

```
if char 1 of card1 = 1 then
go card "Section"&card1 of stack "Academic"
else

if char 1 of card1 = 2 then
go card "Section"&card1 of stack "Campus"
end if

if char 1 of card1 = 3 then
go card "Section"&card1 of stack "Entertainment"
end if

if char 1 of card1 = 4 then
go card "Section"&card1 of stack "Maps"
end if

end if
--end if


--For debugging purposes, place this song in the script at a
--trouble spot to figure out what's going on.
--twinkle, twinkle
--play "boing" tempo 200 "a a e5 e f# f# eh"
--dq d c# c# b4 b aw"
if answer1=0 then
put inactive into result1
--twinkle, twinkle
play "boing" tempo 200 "a a e5 e f# f# eh"
--dq d c# c# b4 b aw"
exit mouseDown
end if

--If the coordinate was within the bounds of the area, but
--was not within a specific building, then nothing is to be
--done.
if i=imax+1 then
put inactive into result1
--As an aid in debugging, uncomment the following lines to
--know or verify why nothing happened.
--doom march
play "boing" tempo 500 "ch. ch c ch. ebh d dh c ch b3 c4w"
exit mouseDown
end if

--Uncomment the following line to help let you know if things
--went as planned.
```

```
--play "boing"
play "boing"

end mouseDown
```

Appendix H
Boolean Operation Routines

H.1 Frame Add

```
Button num 6 ID 18 length 1754 name "frame add"
  --Aaaarrrggghhhh!!! On the disc, frames ordered 71 and 72 are only on
  --one frame so, everything after 72 is a frame ahead ie 73 is actually
  --72nd....so moved everything before 72 up one frame...ie 1 is now 2
  --so....to get proper frame number, this program modified so that it
  --subtracts one off of the prompt "first frame #" and adds that to
  --everything to offset by one.
  on mouseUp
  global u, v, order, frame, order2, frame2
  put 0 into u
  --ask "what is the number of the first still?"
  answer "do you really want to do this?" with "yes" or "cancel"
  if it is empty then exit mouseUp
  else
  --The program can be modified...it can either ask for the first
  --still or the number of the first still, in this case 34973-1=
  --34972, can be entered.
  --put it-1 into v
  put 34972 into v
  -- The following line was added to compensate for disc error
  put v-1 into v
  end if

  if u = 0 then put 1 into u
  --The maximum number of still associated with each card is two,
  --so put them into two variables.
  put item 1 of line u of field "order" into order
  put item 2 of line u of field "order" into order2
  repeat until order is empty
  --If the card does not have a specific still, put "none" in
  --field "frame".
  if order is "none" then
  put "none" into frame
  else
  --Offset the order by the proper amount.
  put order+v into frame
  end if
```

```
--Add the frame number to the frame table.
put frame into item 1 of line u of field "frame"

if order2 > 0 then
put order2+v into frame2
put frame2 into item 2 of line u of field "frame"
end if
put u+1 into u
put item 1 of line u of field "order" into order
put item 2 of line u of field "order" into order2
end repeat
--Make a sound to indicate the program has finished.
play "boing"
end mouseUp
```

H.2 Size of Table

```
Button num 16 ID 32 length 790 name "size of table"
  --This routine simply finds the number of entries in the
  --tables on the card, and puts that number is the fifth
  --line of the information field.
  on mouseUp
  global rect, rect1, rect2, rect3, rect4, lookup, lookup1, lookup2
  global line1, line2, i, size, index, location, imin, imax
  global choice, result1, answer1

  answer "Do you really want to do this??" with "yes" or "NO"
  if it is "NO" then
  exit mouseUp
  else

  --The following lines get the number of lines in the table and put
  --it into line 5 of field map.
  put 1 into size
  put line size of field "rect" into rect

  repeat until rect is empty
  put size+1 into size
  put line size of field "rect" into rect
  end repeat
  put size-1 into size
  put size into line 5 of field "map"
  end if
  end mouseUp
```

H.3 Fixit

```
Button num 17 ID 33 length 1230 name "fixit"
  --This is meant to fix the problem on the videodisk
  --Aaaarrrggghhhh!!! on the disc, frames ordered 71 and 72 are only on
  --one frame so, everything after 72 is a frame ahead ie 73 is actually
  --72nd....so moved everything before 72 up one frame...ie 1 is now 2
  --so....to get proper frame number, this program modified so that it
  --subtracts one off of the prompt "first frame #" and adds that to
  --everything to offset by one.
  on mouseUp
  global a, f1, f2
  put 1 into a
  put item 1 of line a of field "order" into f1
  put item 2 of line a of field "order" into f2
  repeat until f1 is empty
  if f1 > 0 then
  --If the frame order is greater than zero but
  --less than 72, add one to properly offset it.
  if f1 < 72 then
  put f1+1 into f1
  put f1 into item 1 of line a of field "order"
  end if
  end if

  if f2 > 0 then
  --If the frame order is greater than zero but
  --less than 72, add one to properly offset it.
  if f2 < 72 then
  put f2+1 into f2
  put f2 into item 2 of line a of field "order"
  end if
  end if
  put a+1 into a
  put item 1 of line a of field "order" into f1
  put item 2 of line a of field "order" into f2
  end repeat
  --Play a sound to indicate the program is finished running.
  play boing
  end mouseUp
```

Appendix I
Rectangular Lookup Routine

```
Button num 15 ID 31 length 3986 name "rect find"
  --The following is a prototype for the lookup program used
  --in the map stack to lookup building locations for various
  --maps. In this version, after the button is clicked, it
  --prompts for a two coordinate number. If this number represents
  --the coordinates within a building's space on the map on the
  --videodisc, then the building is determined and the appropriate card
  --or frame number is grabbed. This program remains useful in
  --debugging problems within a particular map.
  on mouseUp
  global rect, rect1, rect2, rect3, rect4, lookup, lookup1, lookup2
  global line1, line2, i, size, index, location, imin, imax
  global choice, result1, answer1

  ask "what is the 2 coordinate number?"
  if it is empty then exit mouseUp
  else
  --put the coordinate into the variable lookup
  put it into lookup
  --Which info associated with the building do you want
  --grabbed, i.e. the card or frame number.
  ask "which field for line info result1?"
  if it is empty then exit mouseUp
  else
  put it into choice
  end if
  end if

  --The following lines get the number of lines in the table. This
  --function is uneeded in the final version.
  --put 1 into size
  --put line size of field "rect" into rect
  --repeat until rect is empty
  -- put size+1 into size
  -- put line size of field "rect" into rect
  --end repeat
  --put size-1 into size
  --play "boing"
```

```
---Here is the look up function: ****Must get the loc of cd
---window and add to the coordinates before looking up.
put line 5 of field "map" into size
if size is empty then
answer "no table size specified in map field"
play "boing" c6 c7
exit mouseUp
end if
--Put the maximum and minimum line numbers of the filed
--entries into imin and imax
put 1 into imin
put size into imax
put 1 into i
--Put the horizontal coordinates from the field
--"rect" into variables.
put item 1 of lookup into lookup1
put item 2 of lookup into lookup2
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
--Keep repeating while still in the possible range
--of field entries.
repeat while i<=imax
--If the first coordinate is less than the first coordinate
--in the lookup table, there is nothing to be done.
if lookup1<rect1 then
put 0 into answer1
exit repeat

else
--If the first coordinate is between the two horizontal
--coordinates that define the area of a building, put
--the vertical coordinates of the building into variables.
if rect1<=lookup1 and lookup1<=rect3 then

put item 2 of line i of field "rect" into rect2
put item 4 of line i of field "rect" into rect4
--If the vertical coordinate is also between the
--vertical coordinates that define a building,
--the building has been found, so grab the chosen
--information for the same line in the appropriate
--field.
if rect2<=lookup2 and lookup2<=rect4 then
put i into answer1
exit repeat

else
--The vertical coordinate does not lie in the
```

```
--span of the current entries inthe field, so
--go to the next line and try again.
put i+1 into i
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
end if


else
--The horizontal coordinate does not lie in the
--span of the current entries inthe field, so
--go to the next line and try again.
put i+1 into i
put item 1 of line i of field "rect" into rect1
put item 3 of line i of field "rect" into rect3
end if
end if
end repeat

--The following two if statements are used in debugging.
--Different songs are played for different conditions.

if answer1=0 then
put inactive into result1
--twinkle, twinkle
play "boing" tempo 200 "a a e5 e f# f# eh"
--dq d c# c# b4 b aw"
exit mouseUp
end if

--If the coordinate is within the region of the colorspace
--board but not within a defined building space, then the
--doom march is played.
if i=imax+1 then
put inactive into result1
--doom march
play "boing" tempo 500 "ch. ch c ch. ebh d dh c ch b3 c4w"
exit mouseUp
else
put line answer1 of field choice into result1
end if

play "boing"
end mouseUp
```

Appendix J
Sorting Programs

J.1 Card Sort

```
Button num 9 ID 24 length 2226 name "card sort"
  --This program sorts the five fields by the order of the cards.
  on mouseUp
  global try1, try2, name1, name2, l1, l2, rect1, rect2, order1, order2
  global frame1, frame2
  put 1 into l1
  --Put the first line of field "card" into the variable try1 and
  --the second line of field "card" into the variable try2.  Compare
  --the two variables.
  put line l1 of field "card" into try1
  put 2 into l2
  put line l2 of field "card" into try2

  --Repeat until the end of the entries in field "card".
  repeat until try2 is empty
  --If try1 is less than or equal to try to, then move down and
  --compare the next two lines.
  if try1 <= try2 then
  put l1+1 into l1
  put l2+1 into l2
  put line l1 of field "card" into try1
  put line l2 of field "card" into try2
  else
  --If try1 is greater than try2, then grab all of the other
  --four parameters, and switch all five parameters around.
  if try2 < try1 then
  put line l1 of field "name" into name1
  put line l2 of field "name" into name2
  put line l1 of field "rect" into rect1
  put line l2 of field "rect" into rect2
  put line l1 of field "order" into order1
  put line l2 of field "order" into order2
  put line l1 of field "frame" into frame1
  put line l2 of field "frame" into frame2
  put name1 into line l2 of field "name"
  put name2 into line l1 of field "name"
  put try1 into line l2 of field "card"
put try2 into line l1 of field "card"
put rect1 into line l2 of field "rect"
put rect2 into line l1 of field "rect"
put order1 into line l2 of field "order"
put order2 into line l1 of field "order"
put frame1 into line l2 of field "frame"
put frame2 into line l1 of field "frame"
--If the current line is the first, no need to
--go back any further, so compare the next two
--lines.
if l1=1 then
put l1+1 into l1
put l2+1 into l2
put line l1 of field "card" into try1
put line l2 of field "card" into try2
else
--Go back a pair of lines and check to see if
--the switched line is in the proper spot or if
--it needs to be moved further up the list.
put l1-1 into l1
put l2-1 into l2
put line l1 of field "card" into try1
put line l2 of field "card" into try2
end if
end if
end if
end repeat
--The following noise indicates the sort has been completed.
play "boing"
end mouseUp
```

J.2 Rectangular Coordinate Sort

```
Button num 8 ID 23 length 3220 name "rect sort"
  --This program sorts the rectangular coordinates by the upper left
  --horizontal coordinate.
  on mouseUp
  global card1, card2, name1, name2, order1, order2, frame1, frame2
  global try1, try2, a, b, temp1, temp2
  put 1 into a
  put 2 into b
  put 1 into i
  --Put the first coordinate of lines one and two into the variables
  --try1 and try2, respectively.
  put item i of line a of field "rect" into try1
  put item i of line b of field "rect" into try2
  --Repeat until the end of the entries in field "rect".
  repeat until try2 is empty
  --If try1 is less than try2, no switch is necessary, so go
  --to the next two lines to compare them.
  if try1 < try2 then
  put 1 into i
  put a+1 into a
  put b+1 into b
  put item i of line a of field "rect" into try1
  put item i of line b of field "rect" into try2
  --end if
  else

  --If try1 is greater than try2, put all the other four
  --parameters into variable and switch all five parameters
  --around.
  if try2 < try1 then
  --To debug, the following lines are not essential
  --to the program so comment from here*******
  put line a of field "card" into card1
  put line b of field "card" into card2
  put line a of field "name" into name1
  put line b of field "name" into name2
  put line a of field "order" into order1
  put line b of field "order" into order2
  put line a of field "frame" into frame1
  put line b of field "frame" into frame2
  put name1 into line b of field "name"
  put name2 into line a of field "name"
  put card1 into line b of field "card"
  put card2 into line a of field "card"
put order1 into line b of field "order"
put order2 into line a of field "order"
put frame1 into line b of field "frame"
put frame2 into line a of field "frame"
--to here*********
put line a of field "rect" into temp1
put line b of field "rect" into temp2
put temp1 into line b of field "rect"
put temp2 into line a of field "rect"
--If the current lines are the first and second, no need
--to go back any further, so go to the next two lines
--for a comparison.
if a=1 then
put a+1 into a
put b+1 into b
put 1 into i
put item i of line a of field "rect" into try1
put item i of line b of field "rect" into try2
else
--If the lines are not the first two, back up a pair
--of lines to compare the switched line to the previous
--line to see if it needs to be moved further near the
--beginning of the table.
put a-1 into a
put b-1 into b
put 1 into i
put item i of line a of field "rect" into try1
put item i of line b of field "rect" into try2
end if
end if

--If the coordinates of the two lines are equal, compare
--the next set of coordinates, i.e. if the first coordinates
--are equal, compare the second set.
if try1 = try2 then
if i < 4 then
put i+1 into i
put item i of line a of field "rect" into try1
put item i of line b of field "rect" into try2
else
--If the coordinates happen to be equal, which is
--theoretically impossible, just leave them in the
--same places, and move onto the next pair of coordinates.
if i = 4 then
put 1 into i
put a+1 into a
put b+1 into b
put item i of line a of field "rect" into try1
put item i of line b of field "rect" into try2
end if
end if
end if
end if

end repeat
--Play a sound to indicate the sort is over.
play "boing"
end mouseUp
```

J.3 Name Sort

```
Button num 7 ID 21 length 2260 name "name sort"
  --This program sorts the five parameter fields alphabetically
  --by name.
  on mouseUp
  global try1, try2, l1, l2, card1, card2, rect1, rect2, order1, order2
  global frame1, frame2
  put 1 into l1
  --Put line 1 of field "name" into try1 and line 2 into try2.
  put line l1 of field "name" into try1
  put 2 into l2
  put line l2 of field "name" into try2
  --Keep going until the last entry in the "name" field.
  repeat until try2 is empty
  --If try1 is less than or equal to try2, the two are in proper
  --sequence, so move onto the next line pair to compare.
  if try1 <= try2 then
  put l1+1 into l1
  put l2+1 into l2
  put line l1 of field "name" into try1
  put line l2 of field "name" into try2
  else
  --If try1 is greater than try2, load up the other four
  --parameters into variables, and switch the order of all
  --five parameters.
  if try2 < try1 then
  put line l1 of field "card" into card1
  put line l2 of field "card" into card2
  put line l1 of field "rect" into rect1
  put line l2 of field "rect" into rect2
  put line l1 of field "order" into order1
  put line l2 of field "order" into order2
  put line l1 of field "frame" into frame1
  put line l2 of field "frame" into frame2
  put try1 into line l2 of field "name"
  put try2 into line l1 of field "name"
  put card1 into line l2 of field "card"
  put card2 into line l1 of field "card"
  put rect1 into line l2 of field "rect"
  put rect2 into line l1 of field "rect"
  put order1 into line l2 of field "order"
  put order2 into line l1 of field "order"
  put frame1 into line l2 of field "frame"
  put frame2 into line l1 of field "frame"
  --If the two lines of comparison are the first two, there
```

```
--is no reason to back up to do further comparison, so move
--on to the next two lines.
if l1=1 then
put l1+1 into l1
put l2+1 into l2
put line l1 of field "name" into try1
put line l2 of field "name" into try2
else
--If the two lines are not the first two, then go back a
--pair of lines to check the switched line against the one
--before it to see if it needs to be moved further up the
--table.
put l1-1 into l1
put l2-1 into l2
put line l1 of field "name" into try1
put line l2 of field "name" into try2
end if
end if
end if
end repeat
--Make this sound to indicate the sort is over.
play "boing"
end mouseUp
```

J.4 T Rectangular Sort

```
Button num 10 ID 25 length 3460 name "T rsort"
  --This script sorts the rectangular coordinates of the T stops.
  on mouseUp
  global card1, card2, name1, name2, order1, order2, frame1, frame2
  global try1, try2, a, b, temp1, temp2, station1, station2
  put 1 into a
  put 2 into b
  put 1 into i
  --Put the first coordinate of lines one and two into the variables
  --try1 and try2, respectively.
  put item i of line a of field "rect" into try1
  put item i of line b of field "rect" into try2
  --Repeat until the end of the entries in field "rect".
  repeat until try2 is empty
  --If try1 is less than try2, no switch is necessary, so go
  --to the next two lines to compare them.
  if try1 < try2 then
  put 1 into i
  put a+1 into a
  put b+1 into b
  put item i of line a of field "rect" into try1
  put item i of line b of field "rect" into try2
  --end if
  else
  --If try1 is greater than try2, put all the other four
  --parameters into variable and switch all five parameters
  --around.
  if try2 < try1 then
  --To debug, the following lines are not essential
  --to the program so comment from here*******
  put line a of card field "stationmaps" into station1
  put line b of card field "stationmaps" into station2
  put line a of field "name" into name1
  put line b of field "name" into name2
  put line a of field "order" into order1
  put line b of field "order" into order2
  put line a of field "card" into card1
  put line b of field "card" into card2
  put line a of field "frame" into frame1
  put line b of field "frame" into frame2
  put name1 into line b of field "name"
  put name2 into line a of field "name"
  put station1 into line b of card field "stationmaps"
  put station2 into line a of card field "stationmaps"
put order1 into line b of field "order"
put order2 into line a of field "order"
put frame1 into line b of field "frame"
put frame2 into line a of field "frame"
put card1 into line b of field "card"
put card2 into line a of field "card"
--to here*********
put line a of field "rect" into temp1
put line b of field "rect" into temp2
put temp1 into line b of field "rect"
put temp2 into line a of field "rect"
if a=1 then
put temp2 into line a of field "rect"
--If the current lines are the first and second, no need
--to go back any further, so go to the next two lines
--for a comparison.
put a+1 into a
put b+1 into b
put 1 into i
put item i of line a of field "rect" into try1
put item i of line b of field "rect" into try2
else
--If the lines are not the first two, back up a pair
--of lines to compare the switched line to the previous
--line to see if it needs to be moved further near the
--beginning of the table,
put a-1 into a
put b-1 into b
put 1 into i
put item i of line a of field "rect" into try1
put item i of line b of field "rect" into try2
end if
end if

--If the coordinates of the two lines are equal, compare
--the next set of coordinates, i.e. if the first coordinates
--are equal, compare the second set.
if try1 = try2 then
if i < 4 then
put i+1 into i
put item i of line a of field "rect" into try1
put item i of line b of field "rect" into try2
else
--If the coordinates happen to be equal, which is
--theoretically impossible, just leave them in the
--same places, and move onto the next pair of coordinates.
if i = 4 then
```

```
put 1 into i
put a+1 into a
put b+1 into b
put item i of line a of field "rect" into try1
put item i of line b of field "rect" into try2
end if
end if
end if
end if

end repeat
--Play a sound to indicate the sort is over.
play "boing"
end mouseUp
```

\

Appendix K
Automated Fill Programs

K.1 Still Frame Fill

```
Button num 12 ID 27 length 2328 name "stillframefill"
  --This is the program to fill in still frame numbers into line 1 of
  --each hidden link box on the hypercards by going down the list of
  --cards associated with each map
  --It does not itemize...just overwrites, since for cards with
  --more than one picture, the two frames are entered in one line.
  --For most efficient fill, USE THE CARD SORTED TABLES, since for
  --those entries, such as T indicators, there is no unique card and
  --still.

  on mouseUp
  global a, mapname, card1, frame1
  --For updates, it is not necessary to go through the whole
  --list if the only change is near the end.  The next line
  --prompts the user for the line to start the fill on.
  ask "What line should I start on?"
  if it is empty then exit mouseUp
  put it into a
  --Put the name of the map into a variable to the program
  --understands which card to return to!!!
  put line 1 of field "map" into mapname

  put item 1 of line a of field "card" into card1
  put line a of field "frame" into frame1

  repeat unitil card1 is empty

  --If the first char is 0, nothing needs to be done.  If it
  --is 1-4, go to the appropriate stack, then go to the card
  --and fill in the still frame number into line one of field
  --"links".  Then return to do the next entry of the table.

  if char 1 of card1 = 1 then
  go card "Section"&card1 of stack "Academic"
  put frame1 into line 1 of field "links"
  go card "c"&mapname of stack "Map data"
```

```
else
if char 1 of card1 = 2 then
go card "Section"&card1 of stack "Campus"
put frame1 into line 1 of field "links"
go card "c"&mapname of stack "Map data"
end if


if char 1 of card1 = 3 then
go card "Section"&card1 of stack "Entertainment"
put frame1 into line 1 of field "links"
go card "c"&mapname of stack "Map data"
end if

if char 1 of card1 = 4 then
go card "Section"&card1 of stack "Maps"
put frame1 into line 1 of field "links"
go card "c"&mapname of stack "Map data"
end if

end if
--Once back at the card sort card in the map data stack,
--go to the next line to perform the next still frame fill.
put a+1 into a
put item 1 of line a of field "card" into card1
put line a of field "frame" into frame1

end repeat
put a-1 into a
put item 1 of line a of field "card" into card1
put line a of field "frame" into frame1
--Play this noise to indicate the fill is over.
play "boing"
end mouseUp
```

K.2 Map Frame Fill

```
Button num 11 ID 26 length 2454 name "mapframefill"
  --This is the program to fill in map frame numbers into line 2 of each
  --hidden link box on the hypercards by going down the list of
  --cards associated with each map.
  --It does not itemize...just overwrites. It can itemize by
  --uncommenting lines as shown below.
  --For most efficient fill, USE THE CARD SORTED TABLES, since for
  --those entries, such as T indicators, there is no unique card and
  --still.
  on mouseUp
  global a, mapname, card1, i, i2, mapframe
  --For updates, it is not necessary to go through the whole
  --list if the only change is near the end.  The next line
  --prompts the user for the line to start the fill on.
  ask "What line should I start on?"
  if it is empty then exit mouseUp
  put it into a
  --Put the name of the map into a variable to the program
  --understands which card to return to!!!
  put line 1 of field "map" into mapname
  --Put the frame number of the map into the variable "mapframe"
  put line 3 of field "map" into mapframe
  put item 1 of line a of field "card" into card1

  repeat until card1 is empty

  --If the first char is 0, nothing needs to be done.  If it
  --is 1-4, go to the appropriate stack, then go to the card
  --and fill in the map frame number into line two of field
  --"links".  Then return to do the next entry of the table.


  if char 1 of card1 = 1 then
  go card "Section"&card1 of stack "Academic"
  else

  if char 1 of card1 = 2 then
  go card "Section"&card1 of stack "Campus"
  end if

  if char 1 of card1 = 3 then
  go card "Section"&card1 of stack "Entertainment"
  end if
```

```
if char 1 of card1 = 4 then
go card "Section"&card1 of stack "Maps"
end if
end if


--If there is already something in field "links", this program will
--itemize and not overwrite
--To overwrite, comment the lines below
-- put item 1 of line 2 of field "links" into i2
--if i2 is empty then
put mapframe into line 2 of field "links"
go card "c"&mapname of stack "Map data"
-- else
--  put 0 into i

--   repeat until i2 is empty
--    put i+1 into i
--     put item i of line 2 of field "links" into i2
--  end repeat

-- put mapframe into item i of line 2 of field "links"
-- put 1 into i
--go card "c"&mapname of stack "Map data"
--end if

--Once back at the card sort card in the map data stack,
--go to the next line to perform the next still frame fill.
put a+1 into a
put item 1 of line a of field "card" into card1

end repeat
--Make a sound to indicate the fill is over.
play "boing"
end mouseUp
```

K.3 Map Card Fill

```
Button num 13 ID 28 length 2500 name "mapcardfill"
  --This is the program to fill in map card numbers into line 3 of each
  --hidden link box on the hypercards by going down the list of
  --cards associated with each map
  --It does not itemize...just overwrites. It can itemize by
  --uncommenting lines as shown below.
  --For most efficient fill, USE THE CARD SORTED TABLES, since for
  --those entries, such as T indicators, there is no unique card and
  --still.
  on mouseUp
  global a, mapname, card1, cardname, i2, i, fill
  --For updates, it is not necessary to go through the whole
  --list if the only change is near the end.  The next line
  --prompts the user for the line to start the fill on.
  ask "What line should I start on?"
  if it is empty then exit mouseUp
  put it into a
  --Put the name of the map into a variable to the program
  --understands which card to return to!!!
  put line 1 of field "map" into mapname
  put item 1 of line a of field "card" into card1
  --Put the card number of the map into the variable "cardname".
  put line 4 of field "map" into cardname


  repeat until card1 is empty

  --If the first char is 0, nothing needs to be done.  If it
  --is 1-4, go to the appropriate stack, then go to the card
  --and fill in the still frame number into line one of field
  --"links".  Then return to do the next entry of the table.

  if char 1 of card1 = 1 then
  go card "Section"&card1 of stack "Academic"
  else

  if char 1 of card1 = 2 then
  go card "Section"&card1 of stack "Campus"
  end if

  if char 1 of card1 = 3 then
  go card "Section"&card1 of stack "Entertainment"
  end if
```

```
if char 1 of card1 = 4 then
go card "Section"&card1 of stack "Maps"
end if
end if

--If there is already something in field "links", this program will
--itemize and not overwrite
--To itemize, uncomment the lines below
--put item 1 of line 3 of field "links" into i2
--if i2 is empty then
put cardname into line 3 of field "links"
go card "c"&mapname of stack "Map data"
-- else
--put 0 into i
--This is for stills/cards that appear on more than one map
--repeat until i2 is empty
--put i+1 into i
--put item i of line 3 of field "links" into i2
--end repeat

--put cardname into item i of line 3 of field "links"
--put 1 into i
--go card "c"&mapname of stack "Map data"
--end if

--Once back at the card sort card in the map data stack,
--go to the next line to perform the next still frame fill.
put a+1 into a
put item 1 of line a of field "card" into card1

end repeat
--Make a sound to indicate the fill is over.
play "boing"
end mouseUp
```

K.4 T Frame Fill

```
Button num 14 ID 29 length 1672 name "tframefill"
  --This is the program to get the card and frame numbers
  --for each of the maps under each station on the TMAP
  --For each station, a list of the frame numbers
  --and hypercard numbers of the associated maps are
  --filled into to the card and frame fields

  --The numbers are input by hand originally into the
  --individual cards for each "cmap" in the map data section.

  --When a user clicks on the TMAP, the rect will be found and the system
  --will go to the appropriate hypercard card for the map...
  --all that is really needed is the card number since the map card will
  --store the frame number and open the appropriate map

  on mouseUp
  global a, mapname, card1, frame1, linecheck, i
  ask "What line should I start on?"
  if it is empty then exit mouseUp
  put it into a
  put item 1 of line a of cd field "stationmaps" into mapname
  put line a of cd field "stationmaps" into linecheck
  put 1 into i
  --Keep going until the last entry in the table.
  repeat until linecheck is empty

  repeat until mapname is empty

  go card "c"&mapname
  put line 3 of field "map" into frame1
  put line 4 of field "map" into card1
  -- go card rTMAP
  --For testing purposes: use rTMAP2 and comment line above
  go card rTMAP2

  put frame1 into item i of line a of field "frame"
  put card1 into item i of line a of field "card"

  put i+1 into i
  put item i of line a of  cd field "stationmaps" into mapname

  end repeat
  --Go to the next line.
  put a+1 into a
put 1 into i
put item i of line a of cd field "stationmaps" into mapname
put line a of cd field "stationmaps" into mapname
end repeat
--Play a sound to indicate the program has come to an end.
play "boing"
end mouseUp
```

References

[Altman 88] Altman, China.
Davenport Wants You as Film Pilot, Not Passenger.
Tech Talk , March, 1988.

[Basin 67] Basin, Andre.
The Myth of Total Cinema.
University of California Press, Berkeley, CA, 1967.

[Brondmo, Davenport 89]
Brondmo, Hans Peter and Davenport, Glorianna.
Creating and Viewing the Elastic Charles - a Hypermedia Journal.
May, 1989.
MIT Media Laboratory.

[Cook 81] Cook, David.
A History of Narrative Film.
W W Norton and Co., New York, 1981.

[Davenport 87] Davenport, Glorianna.
New Orleans in Transition, 1983 - 1986: The Interactive Delivery of a Cinematic Case Study.
August, 1987.
.

[Fiderio 88] Fiderio, Janet.
A Grand Vision.
BYTE , October, 1988.

[Mass Micro 88] *ColorSpaceII - Programmer's Reference*
Pre-Production edition, Mass Micro Systems, 1988.

[Multimedia Lab 88]
The Multimedia Lab, Apple Computer, Inc.
Multimedia Production: A Set of Three Reports.
Technical Report 14, Apple Computer, Inc., November, 1988.

[Negroponte 79] Negroponte, Nicholas.
The Impact of Optical Videodiscs on Filmaking.
June, 1979.
.